IMPROVING THE CYBERSECURITY OF CYBER-PHYSICAL SYSTEMS THROUGH BEHAVIORAL GAME THEORY AND MODEL CHECKING IN PRACTICE AND IN EDUCATION

DISSERTATION

Seth T. Hamman

AFIT-ENG-DS-16-S-010

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-DS-16-S-010

IMPROVING THE CYBERSECURITY OF CYBER-PHYSICAL SYSTEMS THROUGH BEHAVIORAL GAME THEORY AND MODEL CHECKING IN PRACTICE AND IN EDUCATION

DISSERTATION

Presented to the Faculty

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

Seth T. Hamman, BA, MS

September 2016

AFIT-ENG-DS-16-S-010

IMPROVING THE CYBERSECURITY OF CYBER-PHYSICAL SYSTEMS THROUGH BEHAVIORAL GAME THEORY AND MODEL CHECKING IN PRACTICE AND IN EDUCATION

Seth T. Hamman, BA, MS

Committee Membership:

Kenneth M. Hopkinson, PhD
Chairman

Barry E. Mullins, PhD, PE
Member

Michael R. Grimaila, PhD, CISM, CISSP
Member

ADEDJI B. BADIRU, PhD
Dean, Graduate School of Engineering and Management

Abstract

This dissertation presents automated methods based on behavioral game theory and model checking to improve the cybersecurity of cyber-physical systems (CPSs) and advocates teaching certain foundational principles of these methods to cybersecurity students. First, it encodes behavioral game theory's concept of level-*k* reasoning into an integer linear program that models a newly defined security Colonel Blotto game. This approach is designed to achieve an efficient allocation of scarce protection resources by anticipating attack allocations. A human subjects experiment based on a CPS infrastructure demonstrates its effectiveness. Next, it rigorously defines the term *adversarial thinking*, one of cybersecurity education's most important and elusive learning objectives, but for which no proper definition exists. It spells out what it means to "think like a hacker" by examining the characteristic thought processes of hackers through the lens of Sternberg's triarchic theory of intelligence. Next, a classroom experiment demonstrates that teaching basic game theory concepts to cybersecurity students significantly improves their strategic reasoning abilities. Finally, this dissertation applies the SPIN model checker to an electric power protection system and demonstrates a straightforward and effective technique for rigorously characterizing the degree of fault tolerance of complex CPSs, a key step in improving their defensive posture.

AFIT-ENG-DS-16-S-010

To the wife of my youth

Acknowledgments

First, thank you to Dr. Hopkinson, my advisor, for his genuine kindness and for always being willing and available to help me. Also thank you to my committee members for their support, guidance, and feedback. Finally, and most of all, thank you to my family, without whose sacrifice this work would not have been possible.

Seth T. Hamman

Table of Contents

Page

List of Figures

Page

List of Tables

Page

IMPROVING THE CYBERSECURITY OF CYBER-PHYSICAL SYSTEMS THROUGH BEHAVIORAL GAME THEORY AND MODEL CHECKING IN PRACTICE AND IN EDUCATION

I. Introduction

Cyber-physical systems (CPSs) integrate computer processing and physical sensors in a continuous feedback loop to obtain efficient control and oversight over an environment. Because of the economic and societal potential of such systems, large investments are being made worldwide to advance CPS technology. However, due to the physical distribution of their critical components and their intrinsically networked nature, they "introduce safety and reliability requirements qualitatively different from those in general purpose computing" [1].

This dissertation presents automated methods based on behavioral game theory and model checking to improve the cybersecurity of CPSs, and advocates teaching certain foundational principles of these methods to cybersecurity students. For illustrative purposes, it applies its findings to the smart grid, a CPS which is a network of computers and power infrastructure that "enhances customers' and utilities' ability to monitor, control, and predict energy use" [2]. The overarching research questions this dissertation examines are:

> **RQ1**: Can automated reasoning, including model checking and integer linear programs that model game theoretic concepts, be applied to improve the cybersecurity of CPSs? If so, can insights gained from these techniques be effectively imparted to cybersecurity students?

It examines these questions over four specific and distinct research components that comprise Chapters 2-5 of this dissertation. Summaries of these components follow.

Chapter 2 explores the scarce resource allocation problem inherent in protecting CPSs from attack by intelligent adversaries. It poses the following research question:

> **RQ2**: Can the concept of level-*k* reasoning be automated to create CPS defense allocations that counteract human-generated attack allocations?

Specifically, the chapter seeks to derive protection resource allocations optimized to obtain the biggest "bang for the buck." Behavioral game theory's concept of level-*k* reasoning provides the framework for modeling the strategic nature of intelligent attackers and insights into predicting their most likely attack allocations. The approach leverages an integer linear program that "solves" the newly introduced security Colonel Blotto game, which models allocating scarce resources across a CPS's infrastructure, for any level of level-*k* reasoning. The effectiveness of the approach is validated by entering its automated defense allocations into an attack and defend competition conducted with human subjects and based on a published smart grid protection system.

Chapter 3 seeks to position the key insights gained from applying level-*k* reasoning to CPS protection planning in Chapter 2 within an accepted framework for educating the next generation of cybersecurity professionals. It does so by applying cognitive psychology research to the concept of adversarial thinking for cybersecurity. It examines the research question:

> **RQ3**: Can Sternberg's triarchic theory of intelligence provide a paradigm for defining adversarial thinking for cybersecurity that identifies practicable student learning outcomes?

The chapter highlights the fact that working from the simplistic definition that adversarial thinking means "the ability to think like a hacker" makes framing student learning outcomes difficult, and without proper learning outcomes, it is not possible to create appropriate instructional materials. It argues that a better understanding of the concept of adversarial thinking is needed in order to improve this all-important aspect of cybersecurity education. The chapter sheds new light on adversarial thinking by exploring it through the lens of Sternberg's triarchic theory of intelligence. The triarchic theory's division of the intellect into the analytical, creative, and practical components provides a helpful framework for examining the characteristic thought processes of hackers. This exploration produces a novel, multidimensional definition of adversarial thinking that leads naturally to three clearly defined learning outcomes, one of which focuses on developing the strategic reasoning abilities of cybersecurity students.

Based on the new definition of adversarial thinking from Chapter 3, Chapter 4 homes in on the challenge of developing the strategic reasoning abilities of cybersecurity students. It examines the research question:

> **RQ4**: Does learning basic game theory concepts improve a student's ability to anticipate the strategic choices made by other people?

The chapter reiterates how strategic reasoning is an important, but often overlooked, aspect of the practice of cybersecurity. It proposes teaching basic game theory to cybersecurity students to help develop their strategic reasoning abilities. To demonstrate the promise of such an approach, it details a pretest-posttest educational experiment with a control group and an original measurement instrument. Details of the treatment, which consisted of two hours of interactive lectures on both traditional and behavioral game

theory, are also provided. The classroom experiment demonstrates that learning about game theory resulted in a statistically significant improvement in the students' abilities to anticipate the strategic choices made by others. Additionally, the chapter suggests that learning about game theory in a cybersecurity class has the potential to fundamentally alter the way students view the practice of cybersecurity. It may help to orient them around the adversarial conflict that is at the heart of cybersecurity, and this could lead to a more strategic-minded, and therefore better equipped, cybersecurity workforce.

Finally, in Chapter 5, attention is focused on the cybersecurity and reliability challenges posed by CPSs. It explores how the discipline of formal methods—the applied mathematics of design verification—can be applied to rigorously characterize the fault tolerance of CPSs. It examines the research question:

> **RQ5**: Can the SPIN model checker be applied to automate the identification of the degree of fault tolerance of CPSs?

The chapter describes how as distributed, communication-based protection systems (a type of CPS) become more prevalent in the emerging smart grid, the task of critically assessing their reliability has become increasingly challenging due to the complexity of their underlying software. It demonstrates that the discipline of software model checking can be applied to smart grid protection software designs to rigorously assess their fault tolerance. It applies the SPIN model checker (SPIN) to a published wide-area backup protection system (WABPS)—a smart grid technology. The WABPS was specifically architected to be highly reliable under various kinds of common failure scenarios, including mechanical malfunctions, erroneous sensor readings, and communication failures. However, because of its built-in redundancy and decentralized peer-to-peer

design, calculating its precise fault tolerance is non-trivial. The chapter shows how SPIN can be applied to the WABPS's design to brute-force prove the limits of the number and types of failures that can occur while the system remains able to successfully perform its function. The same technique is applicable to a wide variety of CPS software designs, and it provides key insights into understanding the security vulnerabilities of such systems.

In summary, this dissertation examines important research questions involving the cybersecurity of CPSs and has two primary focuses. One is on applying its insights in an automated fashion, which may help lower the barrier to their acceptance by the professional community. The other focus is on adapting its research findings to educational contexts, which will help better equip the next generation of cybersecurity professionals.

II. Applying Behavioral Game Theory to Cyber-Physical Systems Protection Planning[*]

2.1 Introduction

As civilization enters the fourth industrial revolution (Industry 4.0), cyber-physical systems (CPSs) will play an increasingly expanding role in society [3] [4]. Because society's dependence on CPSs is directly proportional to their attractiveness to terrorists and other adversaries who are motivated to inflict maximum harm on their enemies [5], protection planning is a vital aspect of the ongoing operation of any real-world CPS.

Large-scale CPSs pose some unique security challenges because they may be geographically dispersed and located in remote areas where it is difficult to provide physical security [6]. Providing adequate protection resources in such contexts is infeasible due to the large attack surface and the limited availability of man hours and money. Therefore, large-scale CPS protection planning is necessarily an exercise in the allocation of scarce protection resources [7].

Not only are protection resources scarce, but they must be allocated in light of the fact that adversaries are strategic actors. The U.S. Office of Homeland Security warns that adversaries (e.g., terrorists, enemy nation states, etc.) perform reconnaissance and undertake intensive planning before making an attack [8]. Any CPS protection scheme that does not take into account attack scenarios waged by intelligent adversaries is naïve and inadequate.

---

[*] This chapter is based on research that will be published in a chapter in an upcoming book on cyber-physical systems:
S. Hamman, K. Hopkinson L. McCarty, "Applying behavioral game theory to cyber-physical systems protection planning," in *Cyber-physical systems: foundations, principles, and applications*, Elsevier, Academic Press, (projected fall 2016).

The research presented in this chapter lays the foundation for approaching the challenge of CPS protection planning in view of these realities. The approach is to model the protection scenario as a newly formulated security game based on the Colonel Blotto (CB) game from game theory. A security game has been defined as a "game-theoretic model that captures essential characteristics of resource allocation decision making" [9]. Then, the security game is "solved" by applying the concept of level-*k* reasoning from behavioral game theory. Behavioral game theory makes it possible to model the strategic nature of intelligent adversaries and provides insights into anticipating and countering their most likely actions. The goal of this approach is to neither over- nor under-protect the critical sites in a CPS infrastructure, thereby achieving the biggest "bang for the buck."

Furthermore, much of the human element involved in protection planning is eliminated by leveraging a mathematical programming solver to determine level-*k* solutions to an integer linear program (ILP) which models the security CB game. The solver outputs a precise allocation of protection resources across critical sites. The ILP is applicable to any size CPS and any amount of protection resources.

In order to provide a clear illustration of how the methodology can be applied to a real-world CPS, this chapter demonstrates how it would allocate protection resources across a notional smart grid special protection system (SPS).

Lastly, the approach is validated by conducting an attack competition with human subjects based on the parameters of the SPS. Using human subjects is the only legitimate way to validate the effectiveness of an approach that has as its goal the countering of attacks waged by intelligent adversaries. Other ways to measure effectiveness, including

performing computer simulations, generating random attacks, or even constructing mathematically rigorous models, fall short because it cannot be convincingly demonstrated that they fully capture the intelligence and strategic nature of human beings.

2.2 Related Work

This work is related to other research efforts that attempt to allocate scarce protection resources effectively over CPSs. [10] promotes building attack trees to find the most damaging attack paths, thereby identifying where protection resources are needed most. Similarly, [7] recognizes the impossibility of protecting every aspect of a CPS infrastructure. It introduces an integrated methodology to prioritize security requirements with the goal of ensuring that the most important tasks are addressed first, instead of proceeding in an ad hoc manner such as "easiest first" or "least expensive first."

Other work has been done regarding the use of game theory in CPS protection planning. [11] emphasizes the need for estimation algorithms that capture realistic attack models, and suggests that game theoretic techniques for modeling rational adversaries may be useful for this task. [12] uses game theory to model the probabilities of successful attacks as a function of the number of components that are attacked and defended. [13] attempts to find the Nash Equilibria for a game theoretic formulation of a CPS security scenario, and it distinguishes between the degradability and the survivability of CPSs after attacks. [14] finds an optimum solution to a CPS security game by utilizing linear programming. [15] incorporates human decision making into a model of defending SCADA control systems by including one level of level-*k* reasoning.

Although not in the context of CPSs, [16] cites successful, real-world implementations of security systems that rely on computer-generated solutions to security games. Its algorithms are currently being used by the Los Angeles International Airport and the U.S. Coast Guard, among others, to derive inspection schedules.

2.3 Approach

When trying to defend a large-scale distributed CPS, protection planners are faced with the dilemma of allocating limited protection resources (e.g., man hours and money) as efficiently as possible over multiple vulnerable sites. If the sites are not all equally valuable, it does not make sense to allocate the resources evenly over the sites (the *equal allocation strategy*). A much better strategy would be to allocate the resources proportionately according to the relative values of the sites (the *proportional allocation strategy*). However, these two approaches, like any plain optimization formulation, fail to capture the effects of the attacker behavior in the model [17]. Consequently, these two natural approaches to the scarce resource allocation problem are inadequate.

The approach taken in this chapter to large-scale CPS protection planning is to try and anticipate how an adversary would allocate his attack resources and then to deploy defensive resources accordingly. To accomplish this, the concept of level-*k* reasoning from behavioral game theory is leveraged.

2.3.1 Level-k Reasoning

When engaging in a strategic contest, the first step a person typically employs in formulating a strategy is to make an educated guess as to what his opponent will do. From that point, he can arrange his strategy to beat his opponent's putative strategy. Of course, he may realize his opponent is also rational and is likely following a similar

procedure. This might lead him to try and beat the strategy that he imagines his opponent is going to use to try and beat his initial strategy. This type of back-and-forth reasoning could theoretically continue indefinitely. Behavioral game theorists, who have extensively studied the dynamics of human beings engaged in strategic interactions, have termed this thought process *level*-k *reasoning* [18].

In the concept of level-*k* reasoning, the natural, instinctual strategy is denoted as the level-0 (L0) strategy, the first logical extension of it as the L1 strategy, then L2, and so on. To summarize, the L*k* type assumes his opponent is an L(*k*-1) type.

Over decades and in many contexts, behavioral game theorists have empirically studied how many levels deep people typically descend into the level-*k* reasoning process. One noteworthy attempt to isolate the level-*k* reasoning process from possible confounding variables is the 11-20 money request game [19]. In this game, two participants, independently of one another, are asked to choose an amount of money between $11 and $20, and they are told they will be given whatever amount of money they choose. Additionally, they are told that they will earn a $20 bonus if they choose exactly $1 less than the other participant.

The L0 strategy in this game is to ask for $20—it is the instinctual starting point since it is the highest amount of money available. From there, the L1 strategy is to ask for $19 in anticipation of the other participant asking for $20, because this will result in the $20 bonus. The L2 strategy is to ask for $18, and the L3 strategy is to ask for $17. Around 80% of the subjects in a study conducted with 108 participants chose between $17 and $20, and the authors demonstrate that these choices represent between three and zero levels of level-*k* reasoning, respectively.

This leaves the other 20% of participants who chose between \$11 and \$16. Did they employ more in-depth reasoning than the other 80% of participants? Based on ex post interviews with the subjects, the answer to this question is a definitive, "No." Indeed, behavioral game theory researchers have repeatedly demonstrated that humans rarely (if ever) continue to four or more levels of reasoning, either because they do not believe their opponents will continue that far, or because they stop when they reach the limit of their mental capacity [19]. People who do not employ level-*k* reasoning typically describe their strategies as being based on "gut instincts," guesses, or intuition. These strategies require very little time to formulate relative to level-*k* reasoning strategies and typically perform poorly in strategic contests.

Numerous level-*k* reasoning studies have been conducted on vastly different pools of people and the results are similar to those of the 11-20 money request game. Researchers have concluded that the majority of people, no matter what their country of origin, level of intelligence, profession, ethnicity, gender, etc., employ between zero to three levels of reasoning [18]. The approach taken in this chapter is to leverage this basic trait of human nature to derive efficient protection resource allocations. However, before level-*k* reasoning can be applied to the scarce resource allocation problem, a formal model of the problem is needed, which the CB game from game theory provides.

2.3.2 The Colonel Blotto Game

Gross and Wagner devised the CB game to capture the strategic dynamics inherent in allocating scarce resources over multiple sites [20]. In the canonical CB game, two colonels, A and B, compete over K independent battlefields of total aggregate value U. The colonels control M and N soldiers, respectively, and they must distribute

them over the K battlefields independently of one another (in the cover of darkness as the eponymous construction goes). Their choices are revealed simultaneously (continuing the illustration, at dawn), and whichever colonel has allocated the most soldiers to a particular battlefield wins that battlefield. Each colonel's goal is to maximize his own utility.

There are many variations of the basic CB game. The colonels may have the same amount of soldiers or different amounts. The values of the battlefields may be homogenous or heterogeneous. The colonels may agree or disagree on the values of the battlefields. There are also different ways to resolve ties, including denoting a default winner in all cases, splitting the utility between the colonels, or not awarding the utility to either colonel.

Arad and Rubinstein have demonstrated that when people play the CB game, they exhibit level-k reasoning [21]. Therefore, this strategic model provides a sound basis for applying level-k reasoning to CPS protection planning.

CPS protection planning is modeled as a specific type of CB game where the defender and the attacker are the two colonels, protection and attack resources are the soldiers, and CPS critical sites are the battlefields. In order to capture the nuanced dynamics of CPS protection planning, the following game variations were selected:

- Both the defender and the attacker are assigned the same number of soldiers, making them equally matched. This makes sense because both the defender and the attacker would naturally allocate resources in proportion to the size and value of the CPS infrastructure.

- Both the defender and the attacker are assigned 100 soldiers. This choice allows for the easy identification of the proportion of resources allocated to each site (i.e., each soldier is 1% of a colonel‘s budget).
- The battlefields’ values are heterogeneous. Based on their location in the infrastructure and their differing responsibilities, the critical sites of any large-scale distributed CPS will have different amounts of utility.
- It is assumed that the attacker and the defender will value the battlefields symmetrically. Because the model is predicated on a well-planned attack, the attacker, having conducted substantial reconnaissance, will have accurate values for the sites.
- In the case of a tie, the battlefield utility is split evenly.

In addition to these variations one tweak must be made to the canonical CB game to transform it into a security game. In protection planning, unlike in the war version of the game, the critical sites are not neutral ground—they are all owned by the defender by default. In the classic CB game, in the scenario where neither the defender nor the attacker allocate resources to a particular battlefield, the battlefield’s value goes un-awarded. However, in the same situation in a security context, the defender would win that site because he owns all of the sites to begin with. Therefore, in what this chapter terms the *security CB game*, un-attacked battlefields are automatically awarded to the defending colonel, even if those sites are not protected.

2.3.3 Calculating Level-k Strategies

An ILP was devised to model the security CB game. Based on the ILP, a mathematical programming solver (e.g., CPLEX) is able to efficiently calculate the best responses to any set of opponent strategies. By bootstrapping the model with the L0 strategy, the ILP can calculate strategies at any depth of level-*k* reasoning by first computing the best response to the L0 strategy (i.e., the L1 strategy), and then the best response to that strategy, and so on, until the desired level is reached.

The L0 strategy in the CB game is the proportional allocation strategy, as demonstrated by Arad and Rubinstein [21]. The proportional allocation strategy for a colonel with N total soldiers, and a game with set K of battlefields and U total utility is calculated as follows:

$$n_j = u_j/U \times N, \forall\, j \in K \qquad (1)$$

Even though this approach allows one to calculate any level of level-*k* reasoning strategy, which level should CPS protection planners select? The answer is that it depends on what level the attackers select. Based on the findings from behavioral game theory described earlier, it can be assumed that they will use between zero and three levels of level-*k* reasoning.

This might make it appear like the L4 strategy would be the best choice, but the evidence from the experiment conducted with human subjects (detailed below) strongly supports the choice of L3 as the correct defensive strategy for the security CB game. This makes sense because as strategies become more distant from lower-level strategies, they over-protect some sites at the expense of others (i.e., they "overthink" the problem). Therefore, the best place to compete is as near as possible to the majority of the

anticipated attacks. The choice of L3 as the best strategy is also consistent with the results of behavioral game theory competitions [22].

The ILP is as follows:

Let P be the set of possible attack strategies and K be the set of battlefields

Let a_{pj} be the number of soldiers placed at battlefield j in attack strategy p, for $1 \leq j \leq |K|$, $1 \leq p \leq |P|$.

Let z_{jn} be a decision variable where

$$z_{jn} = \begin{cases} 1, \text{ defender places exactly } n \text{ soldiers at battlefield } j \\ 0, \text{ otherwise} \end{cases}$$

for $1 \leq j \leq |K|$, $0 \leq n \leq 100$.

Let ε_{pjn} be an indicator variable that is calculated off-line for each $1 \leq p \leq |P|$, $1 \leq j \leq |K|$, $0 \leq n \leq 100$.

For $n \neq 0$, $\varepsilon_{pjn} = \begin{cases} 1, n > a_{pj} \\ 0, n = a_{pj} \\ -1, n < a_{pj} \end{cases}$, so a tie results in 0 points.

For $n = 0$, $\varepsilon_{pj0} = \begin{cases} 1, 0 = a_{pj} \\ -1, 0 < a_{pj} \end{cases}$, so a 0-0 score results in a win for the defender.

Let u_j represent the utility of battlefield j for $1 \leq j \leq |K|$. A model to find an optimal strategy for the defender becomes:

$$\text{maximize} \quad \sum_{p=1}^{|P|} \sum_{j=1}^{|K|} \sum_{n=0}^{100} u_j \varepsilon_{pjn} z_{jn} \qquad (2)$$

$$\text{subject to} \quad \sum_{j=1}^{|K|} \sum_{n=0}^{100} n \cdot z_{jn} = 100$$

$$\sum_{n=0}^{100} z_{jn} = 1 \qquad \forall j$$

$$z_{jn} \in \{0, 1\} \qquad \forall j, n$$

The objective function is the number of wins, minus the number of losses, where ties count as 0 for $n \neq 0$. The first constraint enforces the rule that the attacker and defender each have only 100 soldiers. The second constraint ensures that the defense cannot place two different amounts of soldiers at one battlefield.

2.4 Illustration

In order to illustrate how this approach would allocate scarce protection resources in a real security setting, in this section a specific large-scale distributed CPS is detailed along with a realistic, although hypothetical, attack scenario.

2.4.1 Attacking the Smart Grid

Lloyds imagines a realistic power grid attack scenario conducted by a highly motivated and capable adversary [23]. The report describes a meticulously planned cyber attack involving considerable reconnaissance and effort that hinges on planting malware in smart grid safety control systems. In the scenario proposed, the malware lies dormant until activated during a peak period of electricity demand, at which point it attacks grid components in a coordinated manner and eventually triggers a *cascading blackout*. A cascading blackout is an "avalanche" of power outages that spreads rapidly and uncontrollably over a vast region. To prevent cascading outages from occurring, *load shedding,* the practice of taking "blocks of customers off-line in order to prevent a total collapse of the electric system," is typically performed [24].

The attack scenario in this chapter is based on Lloyds', and is oriented around the SPS described in [25], which is representative of a general, large-scale CPS. As the power grid evolves into the smart grid, SPSs will be increasingly relied upon to help maintain grid stability. SPSs are CPSs made up of communicating nodes located at key

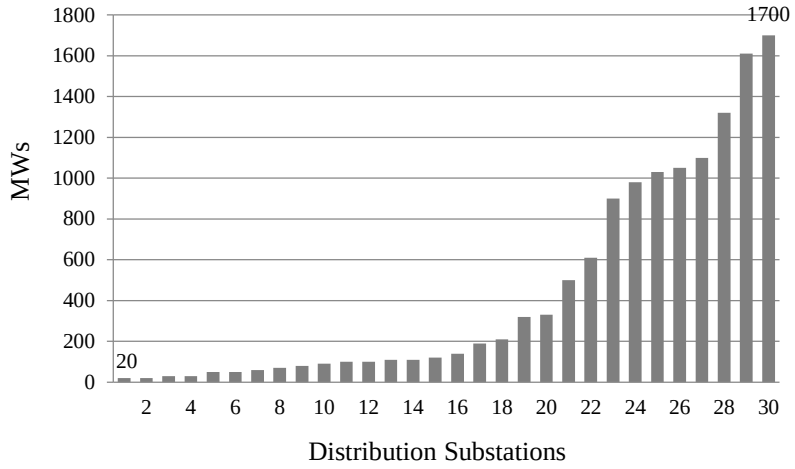

Figure 1. The distribution of 13,030 MWs over the 30 distribution substations in [25]

points in the grid which automatically detect and correct power imbalances in a coordinated manner.

The SPS outlined by [25] is distributed over 30 power distribution substations. The goal of an adversary in the hypothetical attack scenario is to infiltrate the nodes of the SPS and cause them to ignore load shedding commands on demand. Ironically, it is by keeping customers *online* that the adversary hopes to maximize damage. If the adversary can cause a major source of power generation to go offline, perhaps through a physical attack on a key generator, and then prevent mitigating load shedding from taking place, he may be able to create enough of a power imbalance to trigger a cascading blackout.

Because each node manages a different number of megawatts (MWs) (i.e., some distribution substations may be located in urban areas and others in rural areas), they are not all equally attractive to the adversary. His goal is not to obtain control over as many of the 30 nodes as possible, but to gain control over as many MWs as possible. It is

Table 1. The L0-L5 Allocations of 100 Defensive Units Across the 30 Sites

ID	1-7	8-15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
L0	0	1	1	1	2	2	3	4	5	7	8	8	8	8	10	12	13
L1	0	0	0	0	0	3	4	5	6	8	9	9	9	9	11	13	14
L2	0	0	0	0	0	4	0	6	0	9	10	10	10	10	12	14	15
L3	0	0	0	0	2	0	0	0	0	10	11	11	11	11	13	15	16
L4	0	0	0	0	1	0	2	0	2	0	12	12	12	12	14	16	17
L5	0	0	1	2	0	2	2	2	2	0	0	13	13	13	15	17	18

assumed that the adversary, having done considerable reconnaissance, knows the average number of MWs flowing through each distribution substation.

It is also assumed that the adversary does not know how protection resources have been allocated to the nodes. His goal is to allocate more attack resources to particular nodes than the defender has allocated to protecting them. As demonstrated earlier in this chapter, based on findings from behavioral game theory, it is highly likely that the attacker will start with the proportional allocation strategy and then employ between zero and three levels of level-*k* reasoning to allocate his attack resources. This assumption is put to the test in the competition detailed below.

With the infrastructure detailed, the two parameters needed by the mathematical programming solver to calculate level-*k* resource allocations from the ILP have been identified: the number of critical sites (30, based on the 30 distribution substations) and their values (the average number of MWs they control, which are rounded to the nearest 10). This data was compiled from the model power grid in [25] and is shown graphically in Figure 1. The entire power system is comprised of 13,030 MWs, spread out over 30 substations, ranging from 20 to 1,700 MWs each.

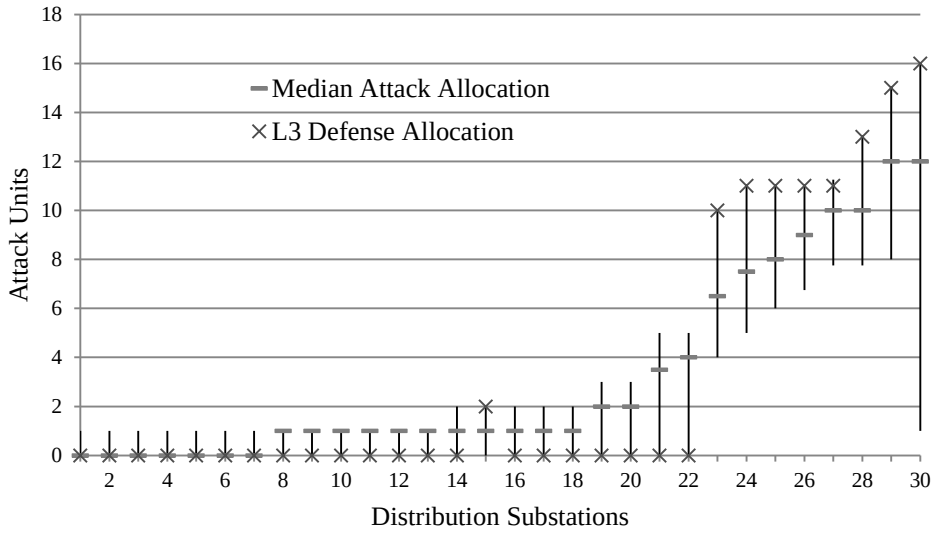

Figure 2. The aggregate human attack strategies (n=92)

Table 1 shows the CPLEX calculated allocations for each of the L0 through L5 strategies. It illustrates that as level-*k* reasoning increases, the trend is to devote more resources to the largest sites at the expense of the lesser sites.

2.5 Validation

To test this approach, an experiment was conducted with human subjects, which is uniquely capable of validating the computer-generated strategy's performance against intelligent human beings.

2.5.1 Experimental Details

Volunteers were solicited from among the engineering and computer science majors at a private Midwestern university. 92 human subjects participated. They competed for gift cards and were given a week to compile their submissions, which incentivized thoughtful participation. The participants were analytically minded, with an

average ACT Math score of 29.83, which exceeds the 93rd percentile. It is believed they are a fair representative sample of intelligent and motivated people in general.

The participants were provided with a prompt based on the SPS defined earlier, outlining the 30 critical sites and their values. There were asked to compete in two different competitions, one as defenders of the infrastructure and one as attackers. (The competitions were subtly different due to the nuance in the security CB game where the defenders own all of the sites by default.) They were tasked with allocating 100 indivisible units of resources across the 30 sites with the goal of winning as much utility as possible. They were clearly informed of all of the specific dynamics of the game, including the equally matched opponent, the rule of the defender winning un-attacked sites by default, and the splitting of utility in case of a tie.

The attack competition served as the basis to measure the defense submissions against. The defense submissions were scored by matching each of them against all of the attack submission in head-to-head contests. The total amount of utility won over all of the head-to-head contests served as the ranking criteria.

The aggregated attack strategies the subjects submitted are shown in Figure 2. The median attack allocations approximately align with the proportional allocation strategy (compare with Figure 1). The vertical bars denote the middle 50% of attack allocations. The L3 defense allocations, marked with X's, defeat approximately 75% of the attack strategies in all nine sites where resources were allocated, which includes the eight most valuable sites. This is a remarkably efficient allocation of defensive resources.

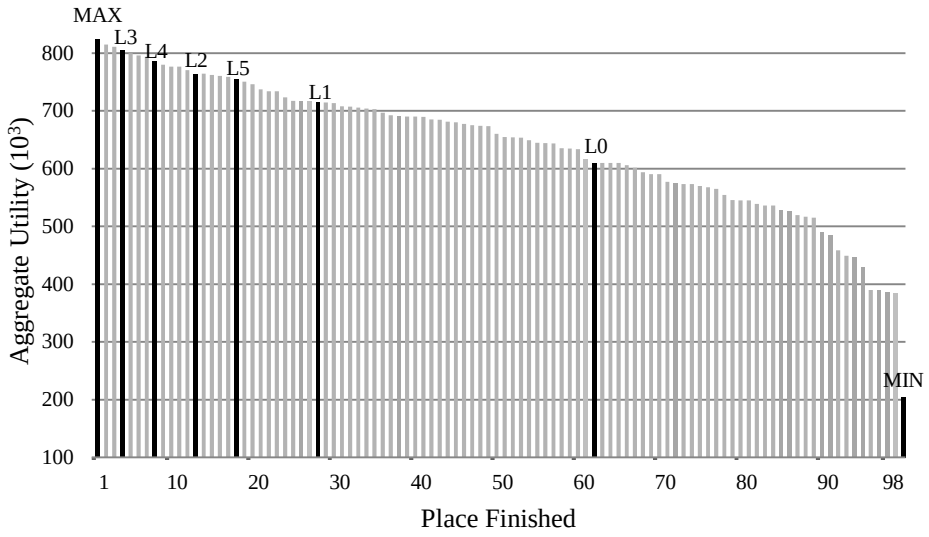

Figure 3. The results of the defense competition (n=98)

2.5.2 Experimental Results

Figure 3 illustrates the results of the defense competition, including the scores of all 92 human participants and the computer-generated L0 to L5 strategies. The L3 strategy performed remarkably well, finishing the best of any of the level-*k* strategies in 3rd place. The L3 strategy achieved 32% more aggregate utility than L0 (i.e., the proportional allocation strategy) and only 2% less than the maximum possible utility.

The L0 strategy finished in a tie for 61st place (three of the 92 human competitors also submitted the L0 strategy). This is an alarming result because it is the natural way that limited protection resources would be allocated across a CPS infrastructure in practice, because it is the most "commonsense" approach. However, it is not a *strategic* approach. The reason it did poorly in the competition is presumably because many of the attackers anticipated this "commonsense" defensive posture and allocated their attack resources accordingly. This result provides support for the notion that the L0 strategy is naïve.

Table 2. Competition Result Details for Select Defense Strategies (n=98)

ID	Place in Competition	Percentage of all Possible Strategies Outperformed
L0	T61	82.2%
L1	26	97.3%
L2	12	99.2%
L3	3	99.7%
L4	7	99.5%
L5	17	99.0%
Human-Best	1	99.8%
Human-Worst	98	10.5%
MAX	N/A	100.0%
MIN	N/A	0.0%

Also noteworthy is that the first three level-*k* strategies did progressively better, peaking at L3, and then began to degrade with the L4 strategy. This is consistent with behavioral game theory findings, and suggests that the L4 and L5 strategies are over-optimized. Meanwhile the L3 strategy competed in the same space as most of the competitors, and likely due to its mathematical precision, out-performed most of the other L3 thinkers in the competition. The experimental results confirm that the L3 strategy achieves the best balance across all sites, neither over- nor under-allocating resources.

To help gain insight into a more absolute measure of performance (as opposed to the relative performance against this specific pool of human competitors), CPLEX was used to solve for the optimal defensive strategy (designated MAX) in the competition. MAX could only be determined *after* the competition because it requires perfect knowledge of all of the attack strategies. (No participant, including the authors, was privy to the specific attack strategies during the competition.) MAX represents the maximum achievable amount of utility.

The minimum achievable amount of utility (designated MIN) was also calculated by entering a strategy with zero resources allocated to all of the sites. MIN achieved some utility because it won sites where no attack resources were allocated.

In this competition, the total number of possible allocations of 100 units of resources over 30 sites equals:

$$\binom{100 + 30 - 1}{30 - 1} \approx 6 \times 10^{28} \qquad (3)$$

This is an enormous number of strategies, of which the competition submissions represent only a minute subset. Assuming that if all possible defensive submissions were to be entered into the competition, they would be roughly normally distributed between MIN and MAX, and by setting MIN and MAX each three standard deviations on either side of the mean, it is possible to use the cumulative distribution function for the standard normal distribution to calculate an absolute percentage score for every strategy in the competition. These values are summarize in Table 2. The level-*k* strategies peak at L3, which finished 3rd place in the competition against 92 human competitors, and outperformed approximately 99.7% of all possible strategies against this set of human attackers.

The L3 strategy performed better than 99.7% of all possible defensive allocations against this pool of attackers. For comparison purposes, L0 performed better than 82.2%, and the last place human finisher in the competition performed better than only 10.5%. On average, the defense strategies submitted into the competition outperformed 89% of all possible defensive allocations, which is not surprising since they were created by

humans. They would naturally outperform the random (i.e., unintelligent) strategies that make up the vast majority of all possible strategies.

2.6 Conclusion

In conclusion, this chapter argues that any promising approach to CPS protection planning *must* take into account the strategic nature of powerful and highly motivated adversaries. Failure to do so is naïve and unlikely to be effective. To accomplish this, behavioral game theory is leveraged, which is a field that has extensively studied and documented how human beings behave in strategic scenarios. To make concrete the connection between CPS security and behavioral game theory, the security CB game was created, which is a rigorous model of the scarce resource allocation problem inherent in defending any large-scale CPS from attack.

Furthermore, by using CPLEX in conjunction with an ILP, the allocation computations are automated. This is helpful because it provides a mathematically sound basis for resolving the trade-offs and difficult decisions inherent in the scarce protective resource allocation problem.

Most importantly, this chapter demonstrates that it is possible to do much better than the obvious, straightforward approach of allocating scarce protection resources proportionately across a CPS infrastructure. The proportional allocation strategy is highly unlikely to effectively counteract an attacker's resource allocations, as demonstrated by the attack competition conducted with human subjects, because it is naïve and not strategic. On the other hand, the computer-generated L3 strategy performed very well in the competition, demonstrating the validity of the overall approach.

The findings from this work are intended to be the *beginning* of CPS protection planning, not the end. In other words, any serious attempt to protect a large-scale CPS will involve the strategic allocation of scarce resources as a starting point. The security CB game and the ILP provide protection planners with data to help them make the difficult decisions inherent in this task. Because of its general nature, this approach is applicable to a wide variety of CPSs situated in various contexts, but it does not provide protection planners with the implementation details they ultimately need.

Future work could explore how the allocation of man hours and money can be made concrete in a specific CPS infrastructure. For example, log auditing and network monitoring are two tasks where man hours are invariably in limited supply. How, therefore, should security personnel divide up these man hours amongst the many nodes that make up a large-scale CPS? This research argues that as opposed to a proportional allocation of man hours, they should choose the mathematically optimized L3 strategy. As another example, [6] notes that security in CPSs must be accomplished in part by making the nodes resilient because they are inherently vulnerable. But resilience is inherently a matter of degree. The security CB game and L3 reasoning can help determine where "extra resilience" should be placed and exactly how much is necessary. For example, if budget is available to purchase additional security measures (e.g., biometric authentication, surveillance cameras, anti-tamper hardware, etc.) for geographically disparate sites in a CPS, this methodology can shed light on how one should choose which sites to upgrade and by how much.

III. Teaching Adversarial Thinking for Cybersecurity[*]

3.1 Introduction

It is widely acknowledged that teaching adversarial thinking to cybersecurity students is important. In a recent editorial highlighting the state of cybersecurity education in colleges and universities, Fred Schneider writes, "Can adversarial thinking for cybersecurity even be taught, or is it an innate skill that only some can develop? The answer, which is neither known nor aggressively being sought by those who study cybersecurity education, *seems central to the development* [emphasis added] of an effective cybersecurity course" [26].

A team of subject matter experts convened by the Association of Computing Machinery (ACM) to identify cybersecurity curricular guidelines agrees that teaching adversarial thinking is vital. Their summary report states, "To protect systems…we need to temporarily adopt the thinking process of the malevolent hacker…Developing this way of thinking *must be part of* [emphasis added]…educating cybersecurity professionals" [27].

While there is a consensus that adversarial thinking should be taught in higher education settings, current cybersecurity curricular guidelines, both from academia and industry, omit this aspect of cybersecurity education. The recent "CS Curricula 2013" [28], which made headlines for its new emphasis on cybersecurity, does not explicitly mention the term *adversarial thinking*, nor does the National Security Agency (NSA) in their National Centers of Academic Excellence (CAE) curricular guidelines [29]. What

[*] This chapter is based on research that will be published in an upcoming conference proceedings:
S. Hamman, K. Hopkinson, "Teaching adversarial thinking for cybersecurity," in *Proceedings of the 20th colloquium for information systems security education*, Philadelphia, 2016.

explains this disconnect between the acknowledged importance of teaching adversarial thinking, and the apparent lack of curriculum support for doing so? The hypothesis of this chapter is that part of the problem is caused by a lack of clarity regarding what it really means to "think like a hacker." A necessary step towards addressing adversarial thinking effectively in the classroom is identifying appropriate student learning outcomes, but this cannot be accomplished without first having a clear understanding of what adversarial thinking really means.

This chapter sets out to define the term *adversarial thinking* by viewing it through the lens of cognitive psychology. As a discipline that studies the human brain, cognitive psychology provides a good foundation for helping to "get inside the minds" of hackers. Specifically, this chapter homes in on Sternberg's Triarchic theory of intelligence as an anchor for understanding how hackers think. Then, with new insights gained from this exploration, a novel, multidimensional definition of adversarial thinking is proposed that leads immediately to three clearly defined learning outcomes and to some new ideas for teaching adversarial thinking to cybersecurity students.

3.2 Background

3.2.1 Hacker Definition

Given the starting point that adversarial thinking means "thinking like a hacker," the first question that should be addressed in any attempt to define adversarial thinking is, "What kind of a hacker?" For example, the following hacker activities differ substantially: email spear phishing, writing worms and viruses, circumventing digital rights management (DRM) protection, coding a buffer overflow attack, and password cracking. Additionally, there are various different broad categories of hackers, ranging

from script kiddies to highly trained professionals, and from insider threats to hacktivists. For the purposes of this chapter, because the emphasis is on exploring adversarial thinking in the context of cybersecurity practice, all references to hackers refer to the individuals whom cybersecurity personnel are hired to prevent from breaking into their networks and computer systems.

3.2.2 Definitions of Adversarial Thinking

To date, there is no commonly accepted definition of adversarial thinking. When the term is used, in many cases it is not defined at all, taking it for granted that adversarial thinking merely means "thinking like your cyber adversary (i.e., a hacker)." However, this raises the obvious question: what is different or unique about the way hackers think?

Two notable attempts to unpack the idea further have been made in recent editorials promoting the teaching of adversarial thinking in cybersecurity education. Melissa Dark, Education Editor for *IEEE Security & Privacy*, proposes the following definition of adversarial thinking: "Let's say that adversarial thinking is the ability to look at system rules and think about how to exploit and subvert them as well as to identify ways to alter the material, cyber, social, and physical operational space" [30]. Another definition comes from Schneider, who writes that adversarial thinking is "the very essence of game theory. In it, actions by each player are completely specified; for cybersecurity and safety-critical systems, identifying possible player actions is part of the central challenge" [26].

On the surface these two definitions are very different, but what they have in common is the identification of some of the salient objects of a hacker's attention. For Dark, these are "system rules" and "operational spaces," and for Schneider it is "player

actions." Hackers undoubtedly bring a unique perspective to system rules, they strive to alter operational spaces to their advantage, and they carefully consider possible player actions. Combining these two definitions in a concise way might lead to the following definition: a*dversarial thinking is the ability to approach system rules, operational spaces, and player actions from a hacker's perspective.*

This is certainly more helpful than the simplistic "thinking like a hacker" definition. However, this chapter takes the exploration a step further in that it orients the term not around the objects of a hacker's focus, but around the primary structures of his intellect. In other words, the goal of this chapter is to provide a more fundamental definition of adversarial thinking that could then, if desired, be applied to various different objects of a hacker's attention, including system rules, operational spaces, and player actions.

3.2.3 Cognitive Psychology

According to the American Psychological Association, cognitive psychology is the study of "higher mental processes such as attention, language use, memory, perception, problem solving, and thinking" [31]. Because of its focus on the human mind, and in particular on the structures of thought, cognitive psychology is a natural place to turn to for guidance in exploring the minds of hackers.

Well-known psychology professor Robert Sternberg proposes a cognitive model called the *Triarchic theory* that breaks the intellect down into three component parts: the analytical, the creative, and the practical [32]. While there are many competing cognitive models, Sternberg's is appreciated for its simplicity and strong explanatory power. Long

Table 3. Summary of Sternberg's Triarchic Theory of Intelligence

Area	Description	Popular Conception	Exemplar
Analytical	Mathematical ability and logical reasoning	IQ	Einstein
Creative	The ability to make unique connections and see the world in original ways	Creativity	Van Gogh
Practical	The ability to plan, strategize, and accomplish goals	Street smarts	Napoleon

before Sternberg, Aristotle developed a roughly parallel three-pronged model of the intellect, which may have provided some of the inspiration for Sternberg [33].

Sternberg's analytical area captures the popular conception of intelligence, and coincides with the notion of IQ. It includes mathematical ability and logical reasoning. The creative area of the intellect includes the ability to make unique connections and to see the world in original ways. Artists, authors, and musicians excel in this aspect of the intellect. And lastly, practical intelligence includes the ability to plan, strategize, and accomplish goals. CEOs and military leaders typically have high degrees of practical intelligence (see Table 3).

The three areas of the Triarchic theory are meant to capture different modes of intelligence that all human beings possess to a greater or lesser extent. The three areas are not necessarily correlated with one another—a person might be above or below average in any given area independent of the other areas. The model is useful to help explain why some people succeed in some arenas and fail in others. Sternberg notes that some students with high analytical intelligence do very well in the highly structured world of undergraduate education, but they struggle as graduate students because they lack creative and practical intelligence, both of which are paramount for conducting and completing original research [32].

3.3 The Triarchic Theory Applied to Hackers

Applying Sternberg's framework to the minds of hackers provides some valuable insights and a more thorough understanding of what makes their way of thinking unique. This section of the chapter views hacker behavior through each of the three lenses provided by the Triarchic theory, with an emphasis on explaining how each category of the intellect contributes to success in hacking.

3.3.1 Analytical

In the popular culture hackers are typically portrayed as highly intelligent "computer wizards." Hackers in television shows and movies sometimes seem like aliens to those around them because of their uncanny technical abilities. Typically, these characters are irresistibly drawn to computing from their youth. While these portrayals are fictitious, there is support for this popular hacker stereotype; hackers do seem to have an unusual affinity and knack for technology.

Hacking involves detailed knowledge of many highly technical aspects of computing, including computer networking protocols, assembly language programming, and operating systems. In Sternberg's paradigm, this technical knack exhibited by computer hackers ties into the analytical component of their intellect. In this case, their analytical gifts translate into a remarkable facility with computers and technology. Having strong technical abilities is vital to hackers because many kinds of cyber attacks involve overcoming significant technological hurdles. Here are a few examples: to infiltrate a computer network, a hacker may need to construct precisely malformed network packets; to exploit a programming flaw, a hacker may have to tediously code a

buffer overflow attack; and to remain undetected on a system, a hacker might need to modify an operating system's libraries.

One real-life example of a hacker who leveraged his analytical intelligence is Robert Tappan Morris. At the age of 19, Morris published a technical paper on a major vulnerability in a trust protocol used in the BSD Unix operating system [34]. A few years later, freshly graduated from Harvard, he used his advanced understanding of computer networking protocols and operating systems to write a software worm that infiltrated an alarming percentage of the computer systems on the Internet at that time [35]. Another example of a hacker who excelled in this area is Elias Levy (a.k.a. Aleph One) who wrote the seminal paper on buffer overflow attacks [36]. Both of these individuals used their analytical gifts to dissect software and network and security protocols, and this enabled them to identify exploits.

In summary, to think like a hacker in terms of the analytical component of his intellect is to embody his technological capabilities, which includes low-level programming skills and a deep familiarity with operating systems and computer networking protocols.

3.3.2 Creative

Sternberg cites "lack of conventionality" as one of the markers for creative intelligence [32]. This is similar to the way cybersecurity guru Bruce Schneier describes what he calls the "hacker mindset." Schneier writes that a hacker is a person who "discards conventional wisdom," and who by "thinking differently," is able to uncover security vulnerabilities that had not occurred to the system's designers [37]. This aspect

of adversarial thinking may be what Dark is referring to in her definition (quoted above) when she mentions the ability to subvert system rules.

Creativity is at the core of the "hacker mindset." While fiction writers excel at creating original stories that capture the imagination, hackers excel at creating original exploits that bend technology in unexpected ways. Both are manifestations of the same root—they involve seeing the world in a unique way, and the ability "to put old information together in a new way," as Sternberg puts it [32]. While most technologists are concerned with making systems work, hackers are obsessed with pushing the limits of systems and exploring possibilities that many people would never consider. This aspect of hacking is the main connection between the pejorative way the term *hacker* is used today, and the original, complimentary term from a previous era which connoted being a highly skilled programmer.

IP fragmentation attacks provide a good illustration of the way hackers apply their creativity to bend technology and protocols. This class of attacks is where IPv4 packets are intentionally fragmented by hackers for purposes ranging from crashing computers to circumventing firewalls [38]. All computer network students learn that routers are programmed to automatically fragment IPv4 packets that are too large to traverse the next hop link, but the creative and unconventional mind of a hacker realizes that packets could also be fragmented by programmers, intentionally, and in unusual ways. This opens up a world of possible attacks, many of which have exposed unsafe security assumptions made by system designers.

In summary, the creative aspect of adversarial thinking involves embodying the unconventional perspectives of hackers which enable them to manipulate technology in unexpected ways.

3.3.3 Practical

The practical component of Sternberg's Triarchic theory is the aspect of the intellect that involves planning, strategizing, and overcoming obstacles to accomplish goals. While script kiddies are known to indiscriminately fire point-and-click exploits at random in hopes of finding unpatched systems, more highly skilled hackers select targets, conduct reconnaissance, carefully plan their attacks, and meticulously cover their tracks [39]. In general, hackers attempt to use their time and resources wisely, and they strive to outwit security personnel. A researcher who conducted extensive interviews with hackers recorded, "One [hacker] described how he attempted to anticipate the moves of his adversary [i.e., security personnel] by stating, 'how can I predict, how can I anticipate what they're going to do?'" [40]. The researcher concludes that strategizing is an essential aspect of hacking. Schneider, in his definition of adversarial thinking (quoted above), probably has the practical component in mind when he compares adversarial thinking to game theory—the study of strategic reasoning.

A good example of a real-life hacker who excels in the area of practical intelligence is the famous social engineering expert Kevin Mitnick. While Mitnick is undoubtedly very intelligent, his intellectual gifts can be better described as street smarts than book smarts. Mitnick had a knack for thinking on his feet, and he was rarely denied the prizes he sought. During his hacking days, he routinely employed strategic maneuvering to evade detection and capture. For example, during his years on the lam

from the FBI, he routinely hacked into his pursuers' phone lines, voicemails, and email accounts, which enabled him to stay one step ahead of them for years [41]. Interestingly, it was not until the FBI enlisted the help of another hacker, Tsutomu Shimomura, that they finally caught him. Because practical intelligence is associated with success in business, it is no coincidence that Mitnick was able to parlay his hacking infamy into the lucrative career as a cybersecurity consultant that he enjoys today.

In summary, adversarial thinking positioned in the light of the practical component of the intellect is embodying a hacker's ability to think strategically. It is captured in the ways hackers plan their attacks, outmaneuver security personnel, and overcome obstacles.

3.3.4 Summary

Having outlined all three areas of the hacker's intellect separately, it may be helpful to take a real-world example of a cyber attack and see how each of the three aspects contributed to the hacker's success. Clifford Stoll published the first detailed account of a computer hacker in the research literature in 1988 [42]. (He later turned the paper into a bestselling book [43].) Although today's cybercrime is worlds apart from the hacking of the 1980's in terms of motivation, scale, and organization, the fundamental techniques of hacking have not changed.

Stoll describes how his hacker was deeply familiar with the Unix operating system and computer networks in general (on the level of a professional systems and network administrator), and was adept at cracking passwords, writing scripts, and modifying operating system utilities to act as Trojan horses. These strengths can be attributed to the analytical component of the hacker's intellect. Stoll also describes how

Table 4. The Triarchic Theory Applied to Adversarial Thinking for Cybersecurity

Area	Adversarial Thinking Application	Example Attack	Summary
Analytical	Understanding technology at a deep level, including computer networking protocols, programming languages, and operating systems	Buffer Overflow	Technological capabilities
Creative	Identifying unsafe security assumptions through manipulating and stretching technology in unexpected ways	IP Fragmentation	Unconventional perspectives
Practical	Reasoning strategically to plan and execute attacks, evade detection, and overcome obstacles	Trojan Horse	Strategic reasoning

the hacker was able to escalate his privileges on systems from a regular user to root level with Gnu-Emacs, a popular text editor with a built-in mail feature which enabled users to communicate with one another by moving files into each other's home directories. The hacker had the key insight that it was also possible to use the mail utility to move files (like a simple shell script programmed to change user permissions when executed by Cron) into the systems directory. This possibility likely never occurred to the Gnu-Emacs developers because there was no legitimate reason to send "mail" to the systems folder. This insight can be attributed to the creative component of the hacker's intellect. And lastly, the paper describes how the hacker installed backdoors so that he could gain access to systems even after they had been patched, how he modified logs and audit trails to avoid detection, and how he employed many shrewd tactics for identifying new login credentials, including searching in emails and files, installing Trojan horses to capture login attempts, and password cracking and guessing. These strategies can be attributed to the practical component of the hacker's intellect.

This short example illustrates that in the case of a skilled hacker, all aspects of his intellect may contribute to his success. While not all areas are strictly necessary, a hacker without analytical intelligence (i.e., technical expertise) is a nonstarter, one lacking

creative intelligence never discovers novel vulnerabilities and is fully dependent on recycled and likely widely known exploits, and one without practical intelligence has little chance of successfully evading detection or of overcoming obstacles.

3.3.5 Adversarial Thinking Definition

A concise summary of the above exploration leads to the following multidimensional definition of adversarial thinking: *adversarial thinking is the ability to embody the technological capabilities, the unconventional perspectives, and the strategic reasoning of hackers* (see Table 4). The word embody used in the definition is intended to capture the sense in which actors embody the characters they play. It connotes "becoming one" with hackers and seeing the world through their eyes. To the extent that cybersecurity students can acquire this ability, in their future careers they will be able to identify the digital fingerprints of hackers in their systems and compete with them on a level playing field (the analytical component), identify and fix security vulnerabilities before hackers have the opportunity to exploit them (the creative component), and anticipate future attacks, thwart attacks in progress, and help track down hackers (the practical component).

3.4 Adversarial Thinking for Cybersecurity Education

As explained in the introduction of this chapter, the reason for developing a more precise definition of adversarial thinking is to help identify appropriate learning outcomes around which curricula can be built. This section of the chapter briefly examines current educational practices in terms of each of the three dimensions outlined in the definition. For each area, three aspects in particular are addressed:

1. Awareness – how aware is the educational community of the importance of this area?
2. Progress – how well is the educational community currently addressing this area?
3. Potential – how much potential is there for developing students' skills and abilities in this area?

3.4.2 Technological Capabilities

Although it is typically not associated with adversarial thinking, in order to think like a hacker, cybersecurity students must understand a hacker's technological capabilities. This cybersecurity learning objective has been understood for a long time, and teaching students technology and the tricks of the trade is the primary emphasis of cybersecurity education today. For example, the NSA's CAE in Cyber Operations curriculum stresses low level programming, software reverse engineering, operating systems theory, computer networking, and many other highly technical topics [29].

Not only is this area of cybersecurity well established, it is also particularly effective at accomplishing its ends due to the fact that most computer science students (i.e., the typical cybersecurity student) enjoy a knack for technology that is on par with hackers.

3.4.3 Unconventional Perspectives

Because it is widely recognized as being important, helping cybersecurity students develop the unconventional perspectives of hackers is the subject of much active research. One recent innovative approach to achieving this involves encouraging students to cheat on an otherwise impossible-to-pass exam. The authors explain, "For it is only by learning the thought processes of our adversaries that we can hope to unleash

the *creative thinking* [emphasis added] needed to build the best secure systems" [44]. Another cybersecurity educator attempts to teach students this type of creative thinking by assigning hacking labs. He writes, "We find students truly learn when challenged with defeating a computer protocol" [45]. Others have written about how Capture the Flag (CTF) exercises also may contribute to developing this type of creativity [46].

Unlike the technological capability area above, computer science students do not necessarily have strong innate creative abilities. On the contrary, most technically minded people are predominately "left brained," meaning that they resonate with logic, rigidity, and rules to the detriment of "outside-the-box" thinking. Therefore, teaching this aspect of adversarial thinking may prove to be an uphill battle. It is not yet known how effective approaches like the ones mentioned above are at developing cybersecurity students' abilities in this area.

3.4.4 Strategic Reasoning

Unlike the previous two areas, there is very little awareness of the need to teach strategic reasoning to cybersecurity students. One hypothesis for this blind spot is that because cybersecurity education was born out of a technical discipline (i.e., computer science), it has tended to stay revolved around technology to the neglect of the human element inherent in cybersecurity. However, without cyber adversaries, there is no cybersecurity. In fact, at the heart of cybersecurity is an adversarial conflict. At least one educational researcher has noted this weakness in cybersecurity education. He writes, "These topics [i.e., the technical aspects of the curriculum] must be augmented with large doses of ethics, legal studies, behavioral science, and military strategic studies" [47].

Table 5. Summary of Adversarial Thinking Instruction in Cybersecurity Education

Dimension	Learning Outcome	Awareness	Progress	Potential
Technological Capabilities	Understand computer networking protocols, low-level programming languages, and operating systems.	●	●	●
Unconventional Perspectives	Identify unconventional uses of software and protocols that could be exploited as attack vectors by hackers.	●	◒	◒
Strategic Reasoning	Anticipate the strategic actions of hackers, including where, when, and how they might attack, and their tactics for evading detection.	○	○	●

Key: High ● Medium ◒ Low ○

As for potential, this area of adversarial thinking is particularly promising because it is believed that, in general, a person's ability to engage in strategic reasoning can be improved. Colin Camerer, author of the seminal text on behavioral game theory, writes, "Strategic thinking seems to be more like learning to windsurf, ski, or fly an airplane, activities that require people to learn skills which are unnatural but teachable, and less like weight-lifting or dunking a basketball, where performance is constrained by physical limits" [18].

3.5 Recommendations

There are at least three helpful observations that emerge from this brief analysis (see Table 5). First, any attempt to teach adversarial thinking to students with little technical aptitude could prove futile, because in order to understand how hackers think, a student must have some baseline level of innate technical ability. This argues for cybersecurity to continue being taught as a sub-discipline of computer science.

Second, associating what Schneier calls the "hacker mindset" with the creative component of the intellect could lead to novel approaches for teaching the

"unconventional perspectives" of hackers. For example, it may be possible to adapt practices used to stimulate creativity in other disciplines (e.g., creative writing) to cybersecurity education.

Third, the strategic dimension of adversarial thinking is not being adequately addressed in the classroom. This observation has already led to progress in cybersecurity education. The next chapter details an educational experiment that was conducted where basic game theory concepts were taught to cybersecurity students. The results show that learning game theory had a statistically significant impact on the students' abilities to anticipate the strategic actions of others. This study demonstrates that with the proper educational support, students can learn how to better compete in the "battle of wits" that sometimes plays out in the practice cybersecurity.

3.6 Conclusion

In conclusion, by defining more precisely what it means to "think like a hacker," this chapter has shed new light on how adversarial thinking can be addressed in the classroom. Perhaps most beneficial is the realization that strategic reasoning is an important, yet overlooked, aspect of adversarial thinking.

Future work could build on this research by potentially expanding the definition to include other aspects of a hacker's mind, such as his motivations and unique personality traits (see [48]). It would be interesting to study whether these types of insights could also prove beneficial to the practice of cybersecurity.

IV. Teaching Game Theory to Improve Strategic Reasoning in Cybersecurity Students[*]

4.1 Introduction

Cybersecurity expert Ed Skoudis, in his popular textbook on the art of computer hacking, highlights the fact that hackers (i.e., cyber attackers) possess various different levels of ability [39]. On one end of the spectrum are low-skilled script kiddies who deploy point-and-click exploits and hope to compromise unpatched systems. On the other end of the spectrum are highly skilled experts who select targets, conduct reconnaissance, carefully plan their attacks, and meticulously cover their tracks. One such expert hacker described how, when he was preparing to strike, he "attempted to anticipate the moves of his adversary [i.e., security personnel] by stating, 'how can I predict, how can I anticipate what they're going to do?'" [40].

Following best security practices is an adequate defense against script kiddies and other low-skilled hackers, but not against hackers on the higher-skilled end of the spectrum. Cybersecurity personnel must focus on more than just technology and best security practices to stop these types of attackers; they must engage with cyber adversaries on a higher, more strategic level. A good example of this strategic cybersecurity mindset is contained in the first detailed account in the research literature of a cyber attack, where Clifford Stoll describes how he was able to contain and eventually help capture a sophisticated hacker by employing strategic reasoning [42].

[*] This chapter is based on an article that has been submitted to the *IEEE Transactions on Education* journal and is under review: S. Hamman, K. Hopkinson, R. Markham, A. Chaplik, G. Metzler, "Teaching game theory to improve strategic reasoning in cybersecurity students," submitted for publication.

Cybersecurity educational curriculums tend to focus solely on technology and security best practices (see [28] and [29]), and do not address the strategic components inherent in the adversarial conflict of cybersecurity. One educational researcher points out this shortcoming: "These topics [i.e., the technical aspects of the curriculum] must be augmented with large doses of ethics, legal studies, behavioral science, and military strategic studies" [47]. Fred Schneider, a prominent voice in cybersecurity education, also notes that an important aspect of cybersecurity involves identifying the potential strategic actions of attackers. He writes that this is "part of the central challenge" of cybersecurity and teaching it "seems central to the development of an effective cybersecurity course" [26]. The ability to anticipate the where, when, and how of a potential attack, and to shore up defenses accordingly, is a valuable skill in cybersecurity.

As a means to teach strategic reasoning to cybersecurity students, this chapter proposes augmenting traditional cybersecurity curriculums with basic game theory content. To demonstrate the promise of such an approach, a pretest-posttest educational experiment with a control group and an original measurement instrument was conducted. Details of the treatment, which consisted of two hours of interactive lectures on both traditional and behavioral game theory, are provided. The experiment demonstrates that learning about game theory resulted in a statistically significant improvement in the students' abilities to anticipate the strategic choices made by others.

4.2 Background

4.2.1 Game Theory

Game theory is the study of interdependent decision making involving two or more players where each strives to maximize his own utility [49]. Game theory was

Table 6. Analytical and Behavioral Game Theory Comparison

	Analytical Game Theory	Behavioral Game Theory
Method	Deductive	Inductive
Approach	Theoretical	Empirical
History	Established in the 1940's by Morgenstern and Von Neumann	Coined by Camerer in the 2000's; Built on experimental game theory
Provides accurate predictions for…	Many repeated-play games	Many one-shot games
Paradigmatic game	The Prisoner's Dilemma	Nagel's Beauty Contest
Key contribution	Nash Equilibrium	Level-*k* Reasoning

established as a discipline in the 1940's as a means to rigorously analyze the dynamics of market competition. It was founded in the field of economics, but in the last few decades it has become an important sub-discipline in many other fields, including political science, law, biology, and international relations [50].

Behavioral game theory is an empirically based form of game theory that trades analytical game theory's presupposition of player perfect rationality for the experimentally observed rationality of players in actual strategic contests (see Table 6).

One of traditional game theory's most important contributions is the Nash equilibrium, which is a stable condition in a game where no player can unilaterally change his strategy to obtain more utility. One of behavioral game theory's most important contributions is the concept of level-*k* reasoning. Level-*k* reasoning makes rigorous the notion of outwitting one's opponent in a strategic contest. In level-*k* reasoning, the level-0 (L0) strategy is the obvious, instinctual choice, the L1 strategy is expecting your opponent to make the most obvious choice, the L2 strategy is expecting your opponent *to expect you* to make the most obvious choice, etc. The levels proceed *ad infinitum* in theory, but most people stop at between one and three levels of reasoning [18].

Both traditional and behavioral game theory can be described as studies in strategic reasoning.

4.2.2 Teaching Game Theory to Improve Strategic Reasoning

It is believed that a person's ability to engage in strategic reasoning is a skill that can be developed. Colin Camerer, author of the seminal text on behavioral game theory, writes, "Strategic thinking seems to be more like learning to windsurf, ski, or fly an airplane, activities that require people to learn skills which are unnatural but teachable, and less like weight-lifting or dunking a basketball, where performance is constrained by physical limits" [18].

Teaching basic game theory has been used as a means to help improve people's basic strategic thinking abilities. For example, the bestselling book *Co-opetition* teaches basic game theory (no equations or graphs) in order to help business leaders make better strategic decisions [51]. A military researcher affirms that the same kind of approach is effective with military personnel. He writes, "Although one can quickly become bogged down with the mathematics of game theory, a rudimentary understanding of its basic principles can prove quite beneficial to military planners" [52]. Some MBA programs also teach basic game theory to improve the strategic thinking abilities of the future business executives in their programs (see [53] and [54] for two examples).

Because it is empirically based, learning about the concept of level-*k* reasoning and how many levels deep people typically descend can prove especially beneficial for improving a person's strategic thinking abilities. Camerer comments anecdotally that after only an hour of level-*k* reasoning training, research subjects off the street perform better than undergraduate game theory students in strategic contests [18].

Table 7. Game Theory Lecture Topics

Topic	Description
Nagel's Beauty Contest game	In this game the players are asked to guess the number that will be 2/3 of the *average* number guessed by all of the players [78]. Played the game in class with all of the students. The results were tabulated on the spot and then discussed.
Strategic Reasoning	Explained the importance of strategic reasoning for cybersecurity, and how it is an important component of adversarial thinking (see Chapter 3).
Game theory intro	Defined and discussed the history and traditional uses of game theory. Covered the concepts of players, moves, and utility [49].
The Prisoner's Dilemma game and the Nash equilibrium	This game describes a scenario where two suspects are being interrogated separately, and are faced with the dilemma of betraying one another in exchange for a lesser prison sentence versus cooperating with one another and not talking. Explained the methodology used to find the Nash equilibrium, then discussed how doping in sports (e.g., professional cycling) is a real-life prisoner's dilemma game [79].
Real-life game theoretical analysis example	"Solomon's Wise Ruling" (recorded in 1 Kings 3:16-28) is the story of two women who come to King Solomon, each claiming to be the mother of the same baby. To identify the real mother, Solomon rules that the baby shall be cut in two and split between them. Analyzed the scenario using game theory and showed that it predicts the outcome that actually occurred given the women's utility preferences [80].
Behavioral game theory	Defined behavioral game theory and explained the important differences with analytical game theory (see Table 6).
Numb3rs clip	Showed a clip from the television show *Numb3rs* which discusses behavioral game theory and the Hide-and-Seek game [81].
The Hide-and-Seek game	In this game the players are asked to guess in which of four boxes (three of which are identical) that other players have hidden a treasure under. Played the game in class with all of the students. Explained focal point biases and the typical results of the game [82].
The Princess Bride clip	Showed the "Battle of Wits" scene from *The Princess Bride* film to introduce the concept of level-*k* reasoning [84].
Level-*k* reasoning	Discussed the concept of level-*k* reasoning, the definition of L0, and the typical proportions of level-*k* reasoning observed in actual strategic contests by examining the 11-20 Money Request game [19]. Re-examined the in-class Beauty Contest game results.
More game examples	Discussed the Traveler's Dilemma game [83]. Also discussed level-*k* thinking in multiple dimensions with the Colonel Blotto game [21].

4.3 Study Methodology

4.3.1 Study Design

An experiment was designed to answer the following research question: does learning basic game theory concepts improve a student's ability to anticipate the strategic choices made by other people? To answer this research question, a pretest-posttest experiment with a control group was designed.

The research subjects were a representative sample of the students enrolled in computer science major classes in a small, private, Midwestern university. The subjects were male and female freshmen through seniors. The treatment group was comprised of students enrolled in a non-elective, introductory cybersecurity course, whereas the control

group and attack subjects (explained below) were comprised of students enrolled in other non-elective classes within the major.

None of the subjects had previously taken a course in cybersecurity or game theory. All of the subjects participated voluntarily. The study was conducted under the auspices of the university's IRB.

4.3.2 The Treatment

The treatment consisted of two hours of interactive lectures on both traditional and behavioral game theory (see Table 7 for a detailed description of the topics covered). The lectures were augmented with slides, whiteboard diagrams, video clips, and interactive whole-class exercises. The goal of the lectures was to teach basic game theory, including behavioral game theory, with an emphasis on clearly communicating foundational principles and big picture ideas. The primary theme of the instruction was that game theory predicts outcomes by analyzing each player's options in light of all of the other players' options. It was stressed that in this process assumptions must be made about player rationality. Players may not be perfectly rational as analytical game theory presupposes, but rational only to a (sometimes predictable) degree of level-*k* reasoning.

One hour of instruction occurred on Tuesday and one hour on Thursday of the same week, both conducted by the same instructor. The control group received lectures on an unrelated computer science topic by a different instructor.

4.3.3 Measurement Instrument

No suitable instrument was identified to measure a student's ability to anticipate the strategic choices made by others, so a cybersecurity themed instrument was designed

called Data Breach (see Appendix A). Data Breach is a novel, two-player, zero-sum, cybersecurity themed game that combines aspects of the Hide-and-Seek and Colonel Blotto games from game theory. The research subjects took the Data Breach exercise twice, once for the pretest and once for the posttest.

The Data Breach exercise casts subjects in the role of a cybersecurity consultant (the defender) whose job is to help catch an insider threat in an attempt to exfiltrate customer data from a company database. Due to technology constraints that exist within the company's legacy computer systems, the data breach cannot be prevented, but it can be detected after the fact by auditing log files. Therefore, the subjects are asked to strategically allocate a limited number of man hours to the auditing of database log files. There are five log files, one for each day of the week Monday through Friday. The subjects are informed that the number of hours assigned to auditing a particular day's log file corresponds to the perceived likelihood of the insider attacking on that day. For example, allocating 10 hours to Monday's log file indicates a belief that there is a 10% chance of an attack occurring on Monday.

An important detail is that the days of the week have differing amounts of utility. The adversary is motivated to exfiltrate as many records as possible, and the number of records grows linearly throughout the week. Consequently, a successful attack on Monday is worth 1 point (-1 for the defender) whereas a successful attack on Friday is worth 5 points (-5 for the defender). The insider threat's goal is to exfiltrate as much data as possible while minimizing his chances of being detected. Detection results in -10 points for the attacker (10 for the defender).

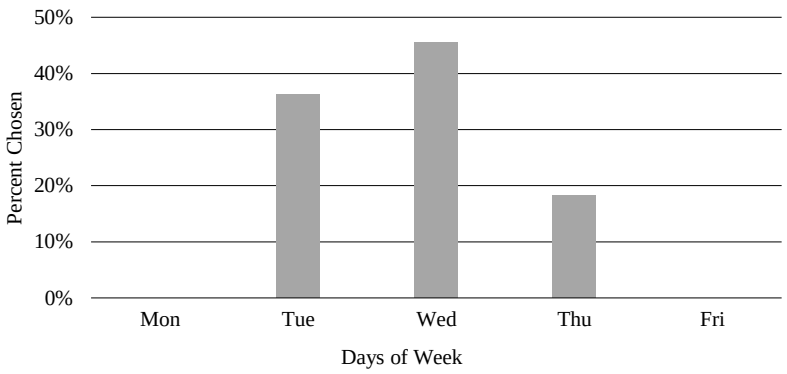

Figure 4. Data Breach aggregated attacks (n=33)

The defenders are tasked with allocating a total of 100 log auditing hours over the five log files. The subjects must assign an integer number of hours from [0, 100] to each day with the constraint that the total number of hours allocated must sum to exactly 100. The defenders were informed that their performance would be measured against the actual days chosen by a peer group of students cast in the role of attackers.

Consequently, it was necessary to collect attack data from a peer group of students. These attack subjects were provided with the same prompt as the defenders, but instead of allocating log auditing hours to the five different log files, they were tasked with selecting one of the days of the week on which to attack. To incentivize thoughtful participation, prizes were offered to the students who identified the best day of the week to attack, as measured against the defenders' allocations of man hours. All 33 students from the peer class participated in the exercise.

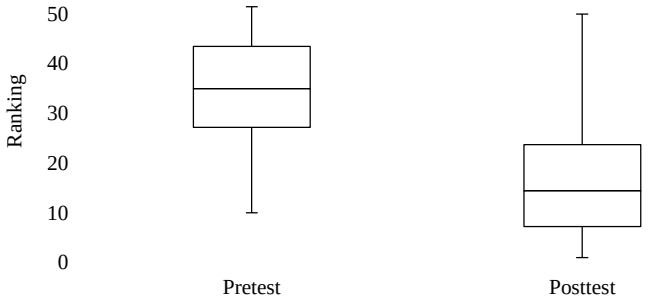

Figure 5. Treatment group pre-post rankings comparison (n=26).

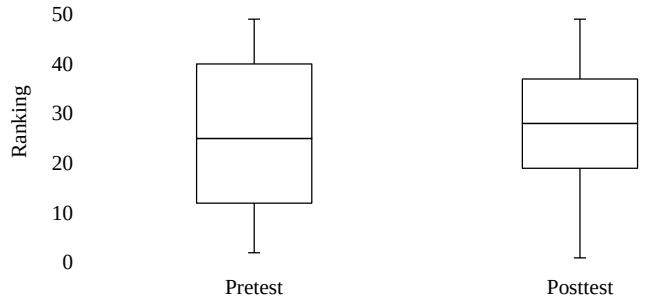

Figure 6. Control group pre-post rankings comparison (n=25).

4.4 Results

4.4.1 Data Analysis

Figure 4 shows the days chosen by the attack subjects in the Data Breach exercise. None of the attackers chose Monday or Friday to attack, and the most popular choice was Wednesday, followed by Tuesday then Thursday.

The Data Breach defender submissions were scored by Microsoft Excel using a formula that directly correlates accurate attack predictions (i.e., hours placed on days on which attacks occurred) with points earned. As an example, *x* hours allocated to Wednesday would earn more points that *x* hours allocated to Thursday because more attackers chose Wednesday than Thursday.

Table 8. Comparisons of Group Performance Rankings

Group (A x B)	A Mdn Rank	B Mdn Rank	n	p-value
Treatment Pre x Post	35	14.5	26	0.041*
Control Pre x Post	25	28	25	0.706*
Treatment Pre x Control Pre	25.75	31	26, 25	0.891**

*Wilcoxon Signed Rank test **Mann-Whitney U test

Because of a floor effect that occurred in the scores of subjects that allocated all of the hours to days on which no attacks occurred, the raw scores did not follow a normal distribution. Therefore, the performance difference between the pretest and posttest was analyzed using the non-parametric Related-Samples Wilcoxon Signed Rank test. The null hypothesis for this test is that the median difference between the pair of observation sets is zero. All statistical analysis was performed with SPSS v.23 using a two-tailed alpha of .05.

26 of 28 treatment group submissions and 25 of 35 control group submissions were included in the analysis (identical pretest and posttest submissions were discarded because they are irrelevant to the Wilcoxon test). Figures 5 and 6 show box-and-whisker plots of the rankings for the pretests and posttests for the treatment and control groups, respectively. (For all of the boxplots, the whiskers indicate the min and max rankings.)

The results of the Wilcoxon Signed Rank test for the treatment group indicate that the null hypothesis should be rejected (p-value = 0.041). This means the subjects' performance improvement in the posttest is statistically significant at the 95% significance level. The results for the control group indicate that the null hypothesis should not be rejected (p-value = 0.706). This means that there was no difference in the median performance (see Table 8).

Table 9. Average Allocation of Hours Across Days

Group	Mon	Tue	Wed	Thu	Fri
Treatment Pre	8.5	11.8	18.4	24.3	36.9
Treatment Post	6.7	14.9	22.4	30.6	25.4
Control Pre	7.6	11.1	16.6	23.5	41.2
Control Post	6.2	9.3	16.1	25.1	43.3
Value of data	1	2	3	4	5

To summarize, the results show that the students were able to more accurately predict the days the attack subjects chose after receiving the game theory treatment. In terms of the research question, this demonstrates that learning about game theory led to an improved ability to anticipate the strategic choices made by others. Because the two groups were similar *except for* the game theory treatment, the performance improvement must be attributed to the treatment (see section 4.4.3 for an analysis of possible confounding variables).

Table 9 details how the groups allocated hours across days. The treatment group redistributed hours on their posttest submissions from the days not chosen by attackers (Monday and Friday) to the days chosen by the attackers (Tuesday, Wednesday, and Thursday). This shows that they anticipated that the attackers would be drawn to the middle of the week.

4.4.2 The Validity and Reliability of the Instrument

The validity of an instrument is a measure of the appropriateness, correctness, meaningfulness, and usefulness of the specific inferences it can help researchers make [55]. The inference the Data Breach exercise was designed to help make is that performance is positively correlated with a subject's ability to predict the strategic choices made by other people. The day chosen by the attackers was a strategic choice on

their part—they were motivated to choose the day which they believed maximized their utility (i.e., the day that resulted in stealing the most data while minimizing the likelihood of being detected). For this reason, and because defender performance is directly proportional to accurate attacker predictions, the Data Breach exercise has high validity.

The reliability of an instrument refers to the consistency of the scores it obtains. Because Data Breach is technically only one question, it is not possible to apply internal-consistency methods such as the Kuder-Richardson approach or Cronbach's alpha to measure its reliability. Therefore, extra care was taken to eliminate threats to its reliability, including the potential for subjects to misunderstand the instructions.

To minimize this threat, the exercise was administered in a quiet classroom setting by an instructor who followed a planned script. The exercise prompt was read out loud to the subjects while they were encouraged to read along. A concise summary of the salient details of the game were reiterated at the end of the prompt to help ensure that all of the subjects clearly understood the rules and the pay offs for both the defender and the attacker. Additionally, a table clearly marked with the values of the days was provided for the subjects to fill in their chosen hour allocations.

4.4.3 Threats to Internal and External Validity

Of the several different threats to the internal validity of an educational research experiment identified in [55], two are relevant to this study: testing and subject characteristics.

With regard to testing, because the subjects took the same measurement instrument twice, a repeat testing effect could account for a performance difference in the

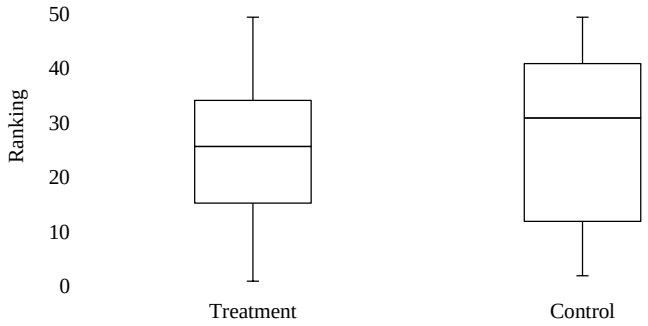

Figure 7. Treatment and control pretests rankings comparison (n=26, 25)

posttest scores. However, the control group showed no performance difference between the pretest and the posttest, which indicates that the repeat testing effect is not a confounding variable in this experiment.

Additionally, it should be noted that caution was taken to ensure that the treatment group did not receive any advantages over the control group that could account for a performance improvement on the posttest (beyond the treatment itself). For example, the Data Breach exercise was not discussed as a type of game theoretical game in the treatment lectures (nor was it discussed at all). Also, the attackers' selections, which were used to gauge the defenders' performance, were not tabulated until after the posttests were completed, and no preexisting empirical results from the Data Breach exercise were available because it is a novel game. Therefore, it was not possible for the instructor to provide any type of performance feedback to the subjects between the pretest and the posttest, either consciously or subconsciously.

With regard to the internal validity threat of subject characteristics, because the control and treatment groups were not randomized, it is possible that the two groups differed in significant ways, and that some of these differences could account for a

performance difference. However, if the control group and the treatment group did differ in ways that could affect the validity of the experiment, this difference should have been reflected in the two group's pretest scores. But as illustrated in Figure 7, the rankings were very similar for the two groups' pretests, and a Mann-Whitney U Independent Samples Median test confirms (p-value = 0.891) that there was no statistical difference in their performance (see Table 8). Additionally, the pretest hour allocations for the two groups were also very similar (see Table 9). The data indicates that the two group's starting points with regard to strategic reasoning were comparable.

Furthermore, a selection bias can be ruled out because both groups of subjects were enrolled in compulsory (i.e., non-elective) computer science classes, and all of the students in both classes participated. Lastly, none of the subjects had ever taken a course in cybersecurity or game theory, which are the two most obvious candidates for characteristics that could impact performance on the Data Breach exercise.

As for the external validity of these results, their scope is limited because the subjects were not a randomized representative sample of cybersecurity students everywhere. Therefore the findings from this experiment can be extended only to students enrolled in computer science classes at small, private, Midwestern universities. However, there is no compelling reason to believe that for any group of people, cybersecurity students or otherwise, learning basic game theory concepts would not result in an improved ability to anticipate the strategic actions of others.

4.5 Discussion

One hypothesis for why the treatment group exhibited an improved ability to predict the days the attack subjects chose is that learning about game theory encouraged

them to consider the perspectives of their adversaries in a new way, and perhaps even for the first time. Because there was a very high correlation between the allocation of hours and the values of the days (see Table 9), it appears that the natural focal point of the students was not on the attackers at all, but on the data they were trying to protect. While it is commonsense to allocate more hours to the more valuable days, this is not a *strategic* way of thinking, because from an attacker's perspective, the obvious choice is to *not attempt* an attack when it is most likely defenders will be expecting an attack. For the attackers, there was actually a negative correlation between the day values and days chosen.

Support for this hypothesis comes from post hoc oral interviews with the treatment subjects. The subjects were asked, "How does knowing about game theory affect your ability to think strategically?" Many students described a newfound awareness of the importance of thinking about *how the adversary is thinking* about a problem. One student described this widely shared revelation bluntly: "It helps you to react better if you are thinking about what the other person is going to be thinking about—how he is going to react to your reactions—[rather] than just assuming that he is going to be a complete idiot." Put in another way, this student is saying that it is natural to fail to take into consideration the perspectives of your adversary when faced with making a strategic decision, and this is equivalent to underestimating his abilities.

One might object that if it was the students' ability to think about the scenario from the attacker's perspective that caused the performance increase on the posttest, then the two hours of game theory instruction may have been superfluous and could have been replaced with a brief exhortation to "try to think like an attacker" while completing the

exercise. There are at least two problems with this view. For one, it underestimates to what extent learning about game theory can help one understand how other people think. It is one thing to *try to think* about how another person would approach a strategic scenario, and another *to be equipped* with the tools to help you do so. And two, providing students with a last minute "hint" may help them do better on an exercise or a test, but it does not demonstrate that the students actually *learned* anything. The intention of education is to help students approach a problem in the correct way *on their own*. The real power of the game theory instruction was that it helped the students *learn* how to think strategically, and this revelation has the potential to make an impact on their ability to practice cybersecurity long after any "hints" would have been forgotten.

4.6 Conclusion

This chapter has argued that strategic reasoning is an important component of cybersecurity, and that one of the goals of cybersecurity education should be to develop the strategic reasoning abilities of students. In Chapter 3, this dissertation has shown that strategic reasoning is actually an overlooked aspect of adversarial thinking (i.e., of "thinking like a hacker")—a widely acknowledged, yet elusive, cybersecurity educational objective. The classroom experiment that was conducted demonstrates that learning about game theory resulted in a statistically significant improvement in the students' abilities to anticipate the strategic choices made by others.

Future research could explore the impact of teaching game theory to cybersecurity students on their future careers, perhaps by conducting a longitudinal study. Learning about game theory in a cybersecurity class has the potential to fundamentally alter the way students view the practice of cybersecurity. It may help to orient them around the

adversarial conflict at the heart of cybersecurity, and this could lead to a more strategic-minded, and therefore better equipped, cybersecurity workforce. As one student reported, "[game theory] is a fascinating topic…The vast majority of the class has focused on how to carry out [cybersecurity] from a technical perspective. Balancing that out with the logic of why and when and where [an attack could] occur is a good [idea]."

V. A Model Checking Approach to Characterizing the Fault Tolerance of Smart Grid Protection Systems*

5.1 Introduction

Smart grid protection systems that utilize communicating processes to provide relays with additional context and to facilitate coordination are potentially far more capable than traditional protection systems, but they also introduce new challenges in critically assessing system reliability. The concurrency that underlies such systems is notoriously difficult to reason about due to the innumerable ways processes can potentially interact and share state.

One common way to test the robustness of smart grid protection systems is by running simulations of basic failure scenarios and then observing the behavior of the system. While helpful, simulations can only go so far in inspiring confidence in the reliability of the systems and their underlying software components. First, protection engineers have to envision the potential failure scenarios ahead of time so they can program them into the simulations, but in many real-world software catastrophes, it is the failure scenarios that the engineers *failed to think of* that end up causing problems, see "Mismatched Assumptions" in [56]. And second, even the most thorough and robust simulation testing can only hope to cover a tiny fraction of the potential failure scenario state space. What is needed is a better way to characterize the entire range of situations where the software can be considered reliable, and to identify with rigor its precise breaking point. This information is invaluable to protection engineers during the

* This chapter is based on an article that has been submitted to the *IEEE Transactions on Power Delivery* journal and is under review: S. Hamman, K. Hopkinson, J. Fadul, "A model checking approach to characterizing the fault tolerance of smart grid protection systems," submitted for publication.

development of new systems, for assessing the quality of competing designs, and for risk management purposes.

Engineers build models of bridges, airplanes, cars, etc. to prove important reliability properties of their designs before they begin construction. Similarly, software model checking tools exist to vet distributed software designs. The aerospace, aeronautical, and automotive industries have used software model checking tools for decades to help validate their safety-critical software systems, see "Logic Model Checking" in [56]. While some references to model checking appear in the power systems literature [57] [58], the practice will be increasingly important as smart grid systems and their software proliferate in power grids.

The SPIN model checker (SPIN) is one of the most popular, easy to use, and mature model checking tools. It was created by Gerard Holzmann, a pioneer in the field of software verification, and currently a Senior Research Scientist for the Jet Propulsion Laboratory for Reliable Software at NASA [59]. He developed SPIN in the 1980's, and he was awarded the prestigious Association of Computing Machinery (ACM) System Software Award for SPIN in 2001 [56].

This chapter advocates for the use of SPIN by the power grid community to test the reliability limits of smart grid protection systems. It makes several contributions. First, it serves as a gentle overview of model checkers for protection engineers motivated by the need to verify increasingly complex smart grid protection systems. Second, it illustrates how out-of-the-box, SPIN can verify that a protection system correctly clears a fault under a given set of conditions. As a final contribution, it demonstrates a straightforward yet elegant technique where SPIN can help characterize the *full* fault

tolerance of a protection system. In other words, SPIN can test how many and what types of system failures can be tolerated, in combination, before the system stops operating properly. This quantitative assessment of the fault tolerance of a communication-based smart grid protection system can be extremely useful when deciding between alternate designs or choosing what additional mechanisms are necessary to ensure proper protection levels. After providing some background on related work and a brief primer on SPIN, this chapter illustrates these techniques by applying SPIN to a published wide-area backup protection system (WABPS).

5.2 Related Work

Petri nets are a graphical and mathematical tool that were designed for modeling complex systems, and they have been used to verify power systems in other research (see [60] [61] [62], for a survey paper, see [63]). Basic petri nets are relatively easy to construct and verify [64], but to model properties of more complex systems like WABPSs, many extensions are likely necessary, including G-nets, colored petri nets, and composite places (see [62] for a verification of a WABPS that utilizes petri net extensions). The complexity these petri net extensions place on software engineers pose a steep learning curve to would-be modelers, and they make it difficult to reason about and verify the correctness of the petri net model itself.

SPIN has many advantages over the petri net approach to smart grid protection system verification. Although model checkers accomplish the same end as petri nets in that they verify software designs, they do so in an automated, brute-force manner, which means it is not necessary for a person to verify the results by following complex logic in a step-by-step manner like in a traditional mathematical proof. SPIN's design description

language is similar enough to other programming languages that proficient programmers can learn to use SPIN relatively easily. And because the models are written using familiar programming constructs, such as data structures, conditional branching statements, and loops, they are easier to comprehend than intricate graph-based petri net models.

Tools similar to SPIN have been applied to verify protection systems. [65] proposes an automated simulation-based verification technique to verify the correctness of relay operations, [57] applies the probabilistic model checker PRISM to verify Markovian models of relay protected components, [58] applies RuleBase, a proprietary IBM model checker, to the verification of hybrid control systems, and [66] applies the Siemens' software tool SIGUARD to verify the protection settings of power systems.

A few of the ways that SPIN has been used successfully in the real world are in verifying NASA mission critical software; such as, the Mars Exploration Rovers and Deep Impact, in a vehicle malfunction investigation involving the 2005 Toyota Camry, and in the verification of medical device transmission protocols [67]. SPIN has also been applied to verify the fault-tolerance of other types of distributed software systems [68] [69].

Many different model checking tools exist, and each has its own set of appropriate verification tasks. For example, software engineers at Amazon apply the model checker TLA+ to the complex distributed systems that underlie their Amazon Web Services [70].

5.3 SPIN Background

SPIN [56] [71] [72] belongs to a class of software tools called model checkers which are a subset of hardware and software verification techniques known as formal

methods—the applied mathematics of design verification. The Federal Aviation Authority (FAA), which has experience investigating the causes of catastrophic software failures in aircraft, recommends, "Formal methods should be part of the education of every computer scientist and software engineer, just as the appropriate branch of applied mathematics is a necessary part of the education of all other engineers" [73]. Model checking is a verification technique coined by Clarke and Emerson in the 1980's [74] that uses optimized algorithms and tailored data structures to efficiently explore all possible system states in a brute-force manner. The theoretical and mathematical foundations of model checking are finite automata theory and linear temporal logic [75].

While neither SPIN nor any model checker should be characterized as the "best" for all tasks and from all perspectives, SPIN has many strengths: it is free and open source, it is very well documented, it is a mature software product, it is under active development as of 2016, the syntax of its PROMELA programming language (a contraction of *Pro*cess *Me*ta*La*nguage) is C-based and familiar, and SPIN has several added-on features to ease model creation (e.g., a graphical user interface and support for auto-generating models from source code).

SPIN was originally an acronym for *S*imple *P*ROMELA *In*terpreter, but has now become a stand-alone term. Because SPIN is intended to model concurrent systems, PROMELA has built-in support for modeling nondeterministic behavior. PROMELA is technically not a programming language like C or Java, but a "systems description language" targeted to "the descriptions of concurrent software systems" [56]. It was designed to help the programmer think in terms of the functions of a distributed system,

```
1  byte x;
2
3  active [3] proctype counting() {
4    if
5    ::(true)-> x = 0;
6    ::(true)-> x = 2;
7    fi
8    printf("Starting value of x: %d\n", x);
9
10   do
11   ::(x < 3) -> x++;
12   ::else -> break;
13   od
14   printf("Ending value of x: %d\n", x);
15
16   assert (x==3)
17 }
```

Figure 8. The PROMELA source code for a simple example program

and it makes it easy to capture common constructs like message passing, shared memory, nondeterministic behavior, and the atomic execution of instruction sequences.

The short example PROMELA program in Figure 8 illustrates a nondeterministic *if* statement on lines 4-7. In PROMELA *if* statements, if more than one guard condition can be evaluated as true (as in this example), then each of them will be executed in some execution of the model, not just the first true guard expression as is the case with *if* statements in traditional programming languages.

The example program also illustrates the ease with which multiple interacting processes can be modeled. The "`[3]`" on line 3 indicates that three concurrent processes will be created, which could easily be changed to any other desired number of concurrent processes. Part of the nondeterminism that SPIN will execute in the model is the arbitrary interleaving of instruction executions by the three processes. Because the variable "`x`" is declared in global scope, all of the processes share its state, so the interleaving of instructions matters.

```
C:\spin>spin promela_example.pml
      Starting value of x: 2
              Starting value of x: 2
          Starting value of x: 2
      Ending value of x: 3
          Ending value of x: 3
              Ending value of x: 3
3 processes created
```

Figure 9. A SPIN simulation run of the example program

```
C:\spin>pan
pan:1: assertion violated (x==3) (at depth 26)
pan: wrote promela_example.pml.trail

(Spin Version 6.4.5 - 1 January 2016)
Warning: Search not completed
        + Partial Order Reduction

Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        acceptance   cycles     - (not selected)
        invalid end states      +

State-vector 24 byte, depth reached 39, errors: 1
       41 states, stored
        0 states, matched
       41 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.002      equivalent memory usage for states
    0.291      actual memory usage for states
   64.000      memory used for hash table (-w24)
    0.343      memory used for DFS stack (-m10000)
   64.539       total actual memory usage



pan: elapsed time 0.016 seconds
pan: rate    2562.5 states/second
```

Figure 10. The SPIN verification run of the example program

SPIN is capable of either interpreting a PROMELA source code model in a simulation run, or of compiling a PROMELA source code model into a standalone C program for verification (the C program is conventionally called “pan” which is short for “protocol analyzer”). When a PROMELA model is run in simulation mode, one particular possible sequence of instructions is selected randomly and then executed.

```
1  C:\spin>spin -t promela_example.pml
2                Starting value of x: 0
3                Ending value of x: 3
4             Starting value of x: 0
5             Ending value of x: 3
6  spin: promela_example.pml:16, Error: assertion violated
7  spin: text of failed assertion: assert((x==3))
8  spin: trail ends after 27 steps
9  #processes: 2
10                  x = 0
11  27:   proc  1 (counting:1) promela_example.pml:17 (state
12  27:   proc  0 (counting:1) promela_example.pml:8 (state
13 3 processes created
```

Figure 11. The "trail" produced by SPIN showing a specific failure scenario

Figure 9 illustrates one possible execution of the example program from Figure 8. The indentation level of the output statements correspond to the process that produced them. In this particular simulation, when all three processes executed the *assert* statement on line 16, the value of x was 3, so no errors were reported.

When a model is run in verification mode, *every* possible sequence of instruction sequences is executed. Figure 10 illustrates that executing the program from Figure 8 in verification mode results in an assertion violation (line 2). When assertion failures occur, SPIN produces a "trail" file (line 3) that can be executed in simulation mode that shows the specific sequence of events that produced the error. Figure 11 is the trail produced by SPIN in Figure 10, and it shows that in this particular failure scenario, process 2 executed the *assert* statement (line 6) in the example program after process 0 set the value of x to 0 (line 12), so x was equal to 0 (line 10) when process 2 asserted it was equal to 3, which caused the error. By default, SPIN stops executing after the first error is found, since one counter-example is sufficient to prove incorrectness. However, SPIN is also capable of enumerating all of the different ways a model can fail validation (in this example there are 510).

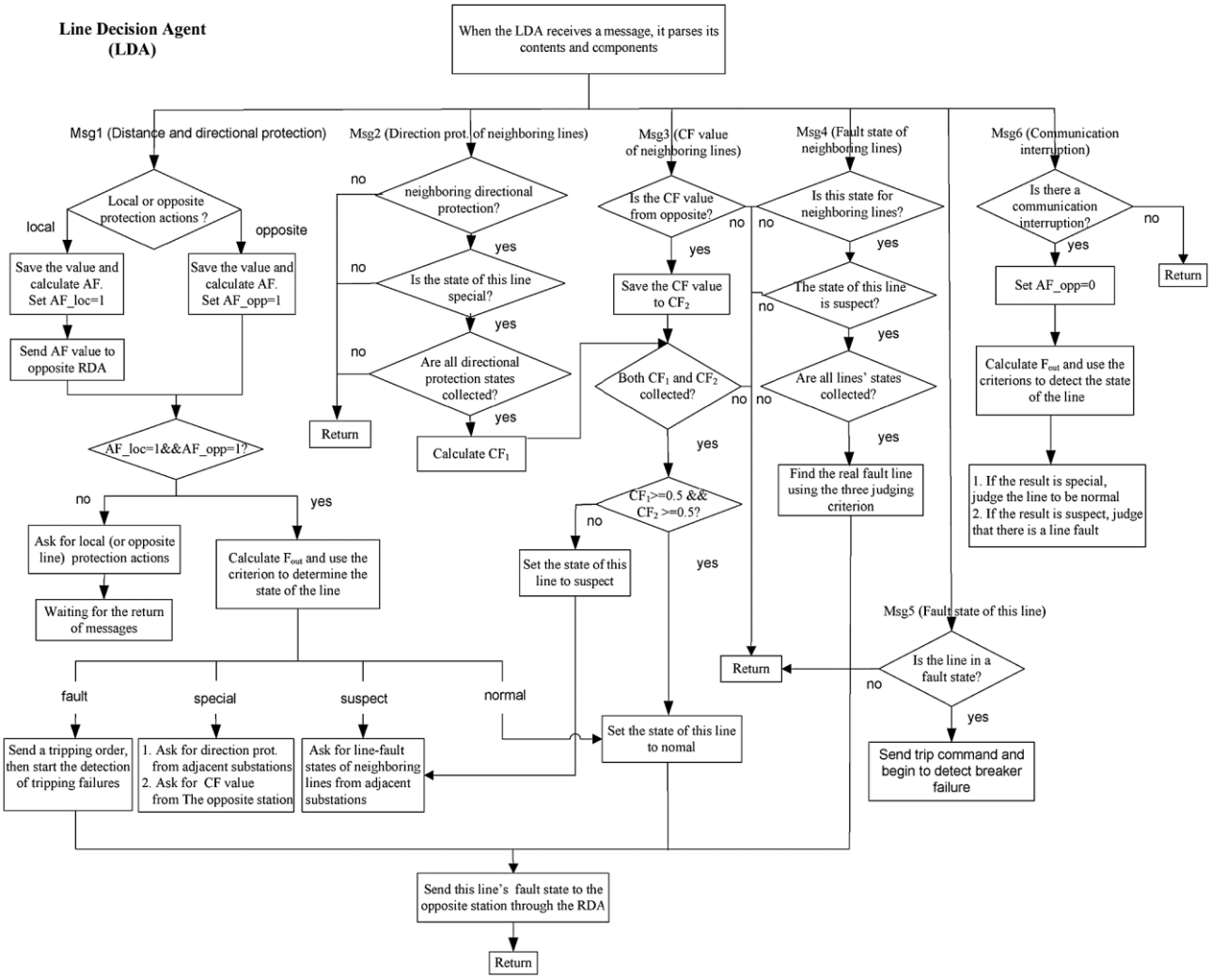

Figure 12. A detailed state transition diagram of LDAs in Tong's WABPS [76]

5.4 SPIN Smart Grid Software Case Study

5.4.1 Tong's WABPS

Because of its elegant and relatively simple design, Tong et al.'s WABPS [76] was selected to illustrate how SPIN can be applied to characterize the fault tolerance of smart grid software. Tong's backup protection system uses smart grid technology to leverage wide-area communication among software decision agents embedded in intelligent electronic devices (IEDs), to clear electrical faults more quickly and efficiently

than is possible with non-communicating protection systems. Smart grid communication allows the agents to gain better situational awareness and facilitates faster and more effective coordination. As [76] describes Tong's WABPS, it "is a regional decentralized peer-to-peer negotiating WABPS multi-agent system that takes into account local and adjacent line, first and second zone, distance protection and directional protection systems as well as fault states from additional lines. This information is then fused to facilitate the creation of a highly accurate WABPS that resides between the main protection and remote backup protection systems."

Figure 12 is a detailed state transition diagram from [76] that illustrates the operation of the local decision agents (LDAs), which is where the core line fault-identifying intelligence resides in the system. During the WABPS's operation, which is triggered by any abnormal state reported by an IED, regional decision agents (RDAs) located in substations alert the appropriate LDAs which then perform a decentralized, three step calculation to ascertain the state of their lines. Over the course of the three steps, an LDA may transition its line's state between *normal*, *special*, *suspect*, and *fault* as it gathers information, performs calculations, shares information with other LDAs, and homes in on its determination of the state of its line. The *special* and *suspect* states are transitional states only, and the *fault* and *normal* states are end states.

In the three step algorithm, first the LDA performs an author-defined Action Factor (AF) calculation based on the state of the directional relay, primary relay, and secondary relay reported by its two local line IEDs (Step 1). Second, depending on the severity of the situation, this may be followed up by an author-defined Certification Factor (CF) calculation that takes into account the state information of IEDs on adjacent

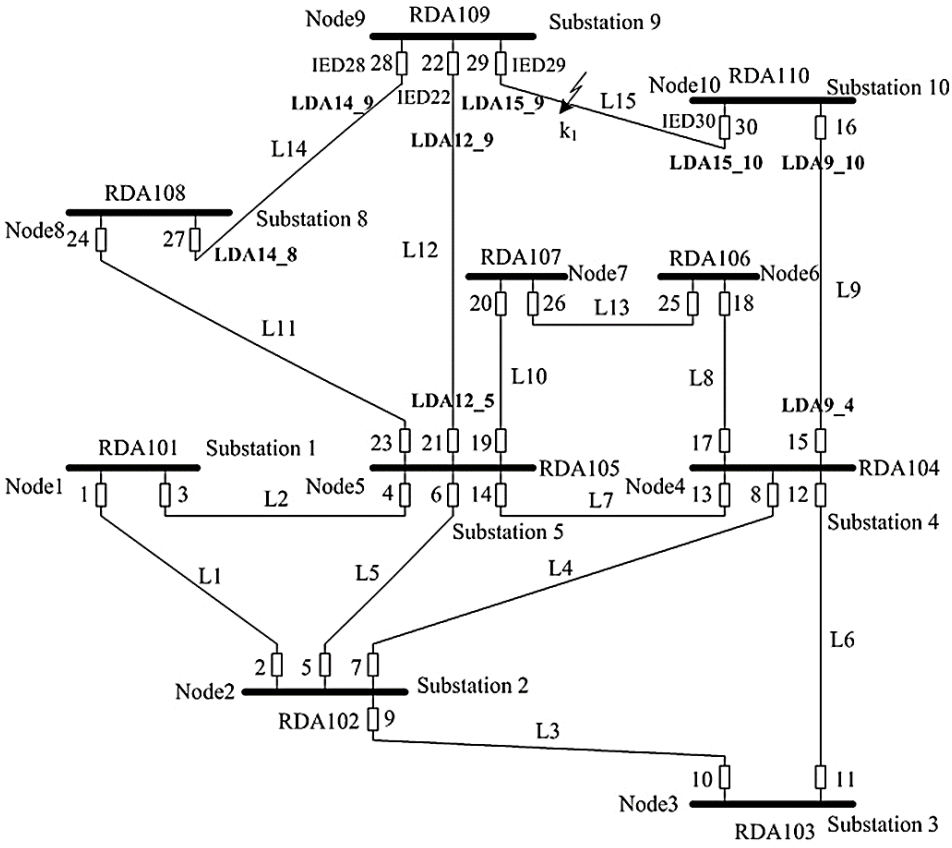

Figure 13. WABPS's layout on the IEEE 14-bus test case [76]

lines (Step 2). Lastly, the LDAs share the state of their line and their CF calculations with neighboring LDAs, and this information is used to resolve any remaining ambiguous line states (Step 3). If after Step 3 an LDA ends up in the *normal* state, no further action is taken. If, on the other hand, an LDA ends up in the *fault* state, a tripping order is sent to the circuit breakers on both ends of the line which are able to mechanically clear the fault.

The main benefit of a wide-area backup protection system is that fault data is collected over multiple lines (i.e., a wide area) and is synthesized to determine the precise location of the fault. In Tong's WABPS, this is done in a decentralized peer-to-peer manner. Decentralized peer-to-peer architectures do not suffer from a single point

Table 10. IEDs Directly Incorporated into LDA Line State Calculations

IED ID	LDA15	LDA12	LDA14	LDA09
3		X		
4		X		
5		X		
6		X		
7				X
8				X
11				X
12				X
13		X		X
14		X		X
15	X			X
16	X			X
17				X
18				X
19		X		
20		X		
21	X	X	X	
22	X	X	X	
23		X	X	
24		X	X	
27	X	X	X	
28	X	X	X	
29	X	X	X	X
30	X	X	X	X

of failure, as centralized architectures do, and this is one of the main features that contribute to their robustness. However, due to the inherent complexity of this type of architecture, it is also non-trivial to reason about how the system behaves when certain failure scenarios occur. Because state is shared among several peer processes, a single IED state change may affect multiple LDAs, and this information may cascade as LDAs coordinate with one another over the course of the algorithm's execution.

The authors of [76] state that the WABPS is "highly reliant and fault tolerant," and they simulate four different failure scenarios on the EPOCHS simulation platform [77] to support their claim. Figure 13 shows the topology of the IEEE 14-bus test system, which is the model power grid that was chosen by the WABPS's authors to test its

operation in the simulations they performed. The four simulations they perform all revolve around the occurrence of a single-line fault on line 15 (L15). L15 was chosen as a representative of a single line fault that could occur anywhere in the system.

For the scenario where a single-line fault occurs on L15, data from 24 IEDs are incorporated into the WABPS's three step algorithm, which is simultaneously executed by LDAs on four different lines (L12, L9, L15, and L14). Each LDA incorporates the state from an overlapping set of IEDs to independently determine the location of the fault (see Table 10 for a summary of how IED state is shared among the LDAs). The core input into the calculation is the state of IEDs, which are complex cyber-physical systems that sense power line data; such as, current, voltage, and frequencies for all three phases of electricity. This information is then used to determine the state of their resident directional relay {bus-to-line fault, line-to-bus fault, no fault}, primary relay {fault, no fault}, and secondary relay {fault, no fault}. During a fault on L15, the WABPS as a whole may incorporate up to a total of 72 discrete states (24 IEDs × 3 relay states per IED) into its calculations. 48 of the states are binary (the primary and secondary relays), and 24 of the states are ternary (the directional relays), making the total state space that is potentially involved with a single-line fault calculation $2^{48} \times 3^{24} \approx 8 \times 10^{25}$ possible states.

However, the vast majority of these states are never reached because the 24 IEDs take on predictable values during a fault on L15. When no errors are present in the system, it is trivial to show that the WABPS performs its function correctly. However, the test of the WABPS's fault tolerance is how well it performs when errors do arise in

the system. The following is the list of possible errors that are anticipated in the WABPS design:

1. an IED takes a bad reading
2. any of the three IED relays fail to operate
3. any of the three IED relays malfunction (e.g., detect a fault when one does not exist or vice versa)
4. an IED transmits a correct relay reading, but the message is corrupted
5. an IED transmits a correct relay reading, but the message is lost
6. an IED fails to operate entirely
7. an IED fails to communicate entirely

The simulations the authors of [76] performed demonstrate that the WABPS functions correctly in four scenarios where different combinations of these seven errors occur.

5.4.2 Modeling Tong's WABPS

Even though SPIN uses advanced algorithms and optimized data structures to achieve extremely high throughput in its brute-force state space search, it is important to keep SPIN models of concurrent software designs as simple as possible, due to the state space explosion problem that quickly arises. For this reason, when constructing a SPIN model, the most important guiding principle is the identification of the *smallest sufficient model* of the software design that captures the properties one wants to prove [56].

Three types of processes are identified in [56] as superfluous in a SPIN model: sink processes, source processes, and filter processes. Tong's WABPS, like almost all real-world software, contains examples of each. The RDAs are filter processes. Their

```
1  failuresSoFar = 0;
2  do
3  ::(failuresSoFar <  maxFailures)->
4    if
5    ::iedIsSelectable(IED03)->failureType(IED03);
6    ::iedIsSelectable(IED04)->failureType(IED04);
7    ::iedIsSelectable(IED05)->failureType(IED05);
8    ::iedIsSelectable(IED06)->failureType(IED06);
9    ::iedIsSelectable(IED07)->failureType(IED07);
10   ::iedIsSelectable(IED08)->failureType(IED08);
11   ::iedIsSelectable(IED11)->failureType(IED11);
12   ::iedIsSelectable(IED12)->failureType(IED12);
13   ::iedIsSelectable(IED13)->failureType(IED13);
14   ::iedIsSelectable(IED14)->failureType(IED14);
15   ::iedIsSelectable(IED15)->failureType(IED15);
16   ::iedIsSelectable(IED16)->failureType(IED16);
17   ::iedIsSelectable(IED17)->failureType(IED17);
18   ::iedIsSelectable(IED18)->failureType(IED18);
19   ::iedIsSelectable(IED19)->failureType(IED19);
20   ::iedIsSelectable(IED20)->failureType(IED20);
21   ::iedIsSelectable(IED21)->failureType(IED21);
22   ::iedIsSelectable(IED22)->failureType(IED22);
23   ::iedIsSelectable(IED23)->failureType(IED23);
24   ::iedIsSelectable(IED24)->failureType(IED24);
25   ::iedIsSelectable(IED27)->failureType(IED27);
26   ::iedIsSelectable(IED28)->failureType(IED28);
27   ::iedIsSelectable(IED29)->failureType(IED29);
28   ::iedIsSelectable(IED30)->failureType(IED30);
29   fi
30   failuresSoFar++;
31 ::else->break;
32 od
```

Figure 14. The nondeterministic *if* statement that verifies all combinations of failures

job is to activate LDAs and coordinate communication between them. The IEDs are source processes. They act solely as sources of relay state information. Lastly, in the WABPS, each IED on a line has an accompanying LDA, which results in two LDA's per line. Because both of the LDA's on a line share all of their state information with one another, one of them can be safely considered a redundant sink process, where information flows in but never out. Therefore, each of these types of processes have been either omitted, or replaced with a higher level abstraction in the SPIN model.

It is important to note that none of these abstractions prevent the SPIN model from capturing important behavior of the RDAs, IEDs, and LDAs in the WABPS. For example, even though RDAs have been abstracted out of the model entirely, it is still possible to model a scenario where an RDA fails by modeling failed communication between LDA processes. Similarly, IEDs in the SPIN model are not represented as independent concurrent processes, but as data structures with three state variables (e.g., directional relay, primary relay, secondary relay). However, by manipulating the state of the IED data structures, it is possible to model IED mechanical and communication failures.

5.4.3 SPIN Testing Tong's WABPS

Using SPIN to characterize the fault tolerance of any smart grid protection system begins with two fundamental observations:

1. The list of things that *could go wrong* (typically captured in the model by nondeterministic *if* statements, as in Figure 14)
2. The list of things that *must go right* (typically captured in the model with one or more *assert* statements)

Out-of-the-box, SPIN will verify the software design by testing every possible combination of items in list 1 against every item in list 2 and will return a binary result: success or failure. However, by parameterizing the list of things that could go wrong with a counter variable, SPIN can be used to test protection system designs against an increasing number of failures until the system reaches its breaking point. When this happens (which for every system is inevitable as the count increases), SPIN can report exactly how many of the combinations lead to assertion failures. With this

information, protection engineers can know where the system can break, why, and to a degree, how likely such a break is in the universe of possibilities.

Tong's WABPS is an experimental, research-only, backup protection system, therefore, a full specification of its operation does not exist. Without a complete specification, it is impossible to fully model the design of a system. In fact, one of the benefits of SPIN promoted by Holzmann in [56] is that SPIN is useful for helping to identify gaps in design specifications, and partly for this reason, it is ideal to incorporate SPIN into the software design phase of the software development life-cycle (SDLC) (this is analogous to the building of model bridges before any actual bridge construction takes place). However, in this chapter, SPIN is being applied after-the-fact to a demonstration-only version of a system.

Therefore, the SPIN model of Tong's WABPS in this chapter was limited to failure scenarios where the system was fully specified, which is the list of seven error types outlined in the previous section during a single-line fault on L15—this is the list of things that *could go wrong*.

The list of things that *must go right* were captured in two different forms. The first is that L15, and only L15, is identified as the faulted line (termed s*trong correctness*). This is the ideal operation of the algorithm and the definition used in [76]. The second is a relaxation where *at least* L15 (as opposed to *only* L15) is identified as the faulted line (termed *weak correctness*). This could also be considered a type of correct operation because the WABPS would still clear the fault, but just not in the most efficient manner possible. Correctness under both definitions were tested in separate rounds of testing.

```
C:\spin>spin wabps_model.pml
     Turned Off IED15
     Turned Off IED30
     Turned Off IED7
         LDA15 Start
         Step 1 - L15 State: SUSPECT
         Step 2 - L15 State: SUSPECT
             LDA12 Start
             Step 1 - L12 State: NORMAL
             Step 2 - L12 State: NORMAL
             Step 3 - L12 State: NORMAL
             LDA12 Final: L15:NO_DATA, L12:NORMAL, L14:NO_DATA, L9:NO_DATA
                LDA14 Start
                Step 1 - L14 State: NORMAL
                Step 2 - L14 State: NORMAL
                Step 3 - L14 State: NORMAL
                LDA14 Final: L15:NO_DATA, L12:NO_DATA, L14:NORMAL, L9:NO_DATA
                    LDA9 Start
                    Step 1 - L9 State: NORMAL
                    Step 2 - L9 State: NORMAL
                    Step 3 - L9 State: NORMAL
                    LDA9 Final: L15:NO_DATA, L12:NO_DATA, L14:NO_DATA, L9:NORMAL
         Step 3 - L15 State: FAULTED
         LDA15 Final: L15:FAULTED, L12:NO_DATA, L14:NO_DATA, L9:NO_DATA
```

Figure 15. Simulation run of WABPS model testing three total IED failures and showing the system functioning correctly

To characterize the full fault tolerance of the system, SPIN was used to stress-test the system in three different ways. First, *n single component IED failures* were tested in combination. Single IED component failures cover items 1-5 on the error list. Second, *m total IED failures* were tested in combination. Total IED failures covers items 6-7 on the error list. Third, combinations of *n* single IED component and *m* total IED failures were tested. These three categories comprehensively cover the entire range of tests from which only a sample of four was selected to simulate in [76]. While simulations are useful for demonstrating that the system *can* operate correctly under a given number of failures, by incrementing the number and type of failures as described above, SPIN can prove the *fundamental fault-tolerance limits* of the system. The result is a rigorous statement of the system's fault tolerance that does not rely upon anecdotal evidence.

Modeling the three step algorithm in SPIN, and the message passing between LDA agents, caused the SPIN model of Tong's WABPS to be somewhat lengthy (around

Table 11. Results of SPIN's Fault Tolerance Verification of Tong's WABPS*

			Strong Correctness		Weak Correctness		Runtime Statistics**	
Type of Failure	Number of Failures	Total Possible Scenarios	Number of Errors	Error Percentage	Number of Errors	Error Percentage	Time Elapsed (secs)	Memory Used (MBs)
Single IED Component	1	96	0	0%	0	0%	0.02	129.7
	2	4,536	0	0%	0	0%	0.96	200.8
	3	140,624	268	0.191%	60	0.043%	60.2	4,359.2
Total IED	1	24	0	0%	0	0%	0.00	128.9
	2	276	0	0%	0	0%	0.05	130.8
	3	2,024	6	0.296%	2	0.099%	0.28	146.7
Single, Total	1, 1	2,208	0	0%	0	0%	0.19	84.9
	2, 1	99,912	247	0.247%	77	0.077%	16.9	1,727.8
	1, 2	25,392	45	0.177%	10	0.039%	2.57	347.4

* All verification runs were performed using SPIN v.6.4.5 (1 Jan 2016) for Linux 64-bit, on a commodity machine with modest specs

** The runtimes and memory usage stats are for the strong correctness runs (weak correctness runs were comparable)

500 lines of code, see Appendix B). Figure 14 is an important code snippet from the model that illustrates the simplicity with which the seven IED failure scenarios from [76] were modeled with SPIN. This simple nondeterministic *if* statement ensures that every possible combination of failures is tested. SPIN even determines whether the order of combinations affects the calculation of faults, so the combinations are tested exactly once. This is a powerful construct, and one of the main ways PROMELA adds value for constructing brute-force verifications compared to trying to leverage traditional programming languages to accomplish the same task. The variable *maxFailures* (line 3) is the counter parameter that was tuned over multiple tests to capture the precise degree of fault tolerance of the system.

Figure 15 shows the output of a simulation run of the SPIN model of Tong's WABPS, where three total IED failures are tested. In this case, IEDs 15, 30, and 7 were selected at random by SPIN, and the results of the calculation shows that the system was able to accurately identify the location of the fault. The output was produced by *printf* statements that are in the model solely for debugging purposes, and have no bearing on

```
C:\spin>spin -t wabps_model.pml
     Turned Off IED29
     Turned Off IED30
     Turned Off IED28
         LDA15 Start
         Step 1 - L15 State: SPECIAL
         Step 2 - L15 State: SUSPECT
             LDA12 Start
             Step 1 - L12 State: NORMAL
             Step 2 - L12 State: NORMAL
             Step 3 - L12 State: NORMAL
             LDA12 Final: L15:NO_DATA, L12:NORMAL, L14:NO_DATA, L9:NO_DATA
                LDA14 Start
                Step 1 - L14 State: SPECIAL
                Step 2 - L14 State: SUSPECT
                    LDA9 Start
                    Step 1 - L9 State: NORMAL
                    Step 2 - L9 State: NORMAL
                    Step 3 - L9 State: NORMAL
                    LDA9 Final: L15:NO_DATA, L12:NO_DATA, L14:NO_DATA, L9:NORMAL
                Step 3 - L14 State: FAULTED
                LDA14 Final: L15:NO_DATA, L12:NO_DATA, L14:FAULTED, L9:NO_DATA
         Step 3 - L15 State: NORMAL
         LDA15 Final: L15:NORMAL, L12:NO_DATA, L14:FAULTED, L9:NO_DATA
spin: wabps_model.pml:219, Error: assertion violated
spin: text of failed assertion: assert(((((agent15Verdict[0]==FAULTED)||(agent12
4Verdict[0]==FAULTED))||(agent09Verdict[0]==FAULTED)))
spin: trail ends after 345 steps
```

Figure 16. The trail simulation run showing one of the two ways that the WABPS fails weak correctness when three total IED failures occur

SPIN verification runs. Of course, in the verification runs, *all* combinations of 3 failures were tested.

5.4.4 SPIN Results

Table 11 provides a summary of the results of the SPIN tests that were performed. The parameters for the number of failures to test were increased until the system reached its breaking point. This resulted in over 275,000 scenarios being tested, and a conclusive statement on the robustness of the WABPS. Even though the robustness of the WABPS was highlighted a number of times in [76], no definitive statements characterizing the limits of its fault tolerance were made. But it can now be stated with rigor that the WABPS's design can handle all combinations of two single IED component failures, all combinations of two total IED failures, and all combinations of one single IED component failure and one total IED failure.

SPIN was also used to calculate the total number of scenarios that caused the WABPS to fail even after it reached its breaking point, which provides more information about the system's robustness. It is beyond the scope of this chapter to qualitatively assess these results, but a protection engineer would be able to incorporate all of this data, along with the probabilities of these types of errors occurring, into a risk management assessment of the system.

It is noteworthy that only a handful of scenarios where three IEDs fail break the system. Figure 16 shows one of the two scenarios that SPIN identified where three total IED failures cause the system to fail to determine that L15 is faulted. As highlighted earlier, the ideal time to incorporate SPIN is early in the SDLC. In this case, this information may have been helpful as the designers of the WABPS were making tweaks to the calculations, parameters, and constants involved in the three step algorithm. SPIN may have been helpful for testing alternatives, and this may have resulted in a modified system with improved fault tolerance.

5.5 Conclusion

In conclusion, this chapter demonstrates how SPIN can be applied to characterize the fault tolerance of smart grid protection systems. It illustrates how the process begins by creating a PROMELA model of the basic design of the system. Next, after the list of things that *must go right* have been incorporated into the model, the type and number of things that *might go wrong* are incremented in separate rounds of testing until an error is reported. SPIN's trail files can then be used to analyze the edge-case failure scenarios.

The SPIN model of Tong's WABPS used in this chapter is a high-level abstraction of the salient design of the system, and is intended only to verify the *design's*

robustness and only under the specified error types. This is analogous to the way that successful simulations of the system, such as those performed in [76], do not guarantee correct real-world operation.

Model checking is not a substitute for other types of system testing, but serves as a helpful complement that is uniquely capable of catching design flaws, identifying incomplete specifications, and characterizing the fault tolerance of systems. SPIN was designed to aid in the development of any type of concurrent software, including the distributed, communication-based systems that are becoming more and more prevalent in the power grid. SPIN and other model checkers have been used for many years to enhance the safety and reliability of critical software systems in many domains. Model checkers can be similarly applied to help mitigate the complexity inherent in safety-critical coordinated network smart grid protection systems.

VI. Conclusion

This dissertation advances automated methods to improve the cybersecurity of CPSs through the application of behavioral game theory and model checking and advocates teaching certain foundational principles of these methods to cybersecurity students. The overarching research questions this dissertation answers are:

> **RQ1**: Can automated reasoning, including model checking and integer linear programs that model game theoretic concepts, be applied to improve the cybersecurity of CPSs? If so, can insights gained from these techniques be effectively imparted to cybersecurity students?

It answers these questions by examining four more specific and distinct research questions that comprise chapters 2-5 of this dissertation. A summary of each chapter's research contributions follows.

Chapter 2 argues that any promising approach to CPS protection planning *must* take into account the strategic nature of adversaries, because failure to do so is naïve and unlikely to be effective. It answers the research question:

> **RQ2**: Can the concept of level-*k* reasoning be automated to create CPS defense allocations that counteract human-generated attack allocations?

To accomplish this, it integrates the concept of level-*k* reasoning from behavioral game theory into an integer linear program that solves the newly defined security Colonel Blotto game, a model of the scarce resource allocation problem inherent in CPS protection planning. It details an experiment performed with human subjects and based on the parameters of a published CPS, where the recommended L3 strategy finished 3rd place out of 92 human competitors. This provides validation that the approach is capable

of automating defense allocations that successfully counteract human-generated attack allocations.

Chapter 3 highlights the need for a proper definition of the term *adversarial thinking* for cybersecurity—arguably cybersecurity education's most important learning objective. Because a robust definition does not exist, it is not clear whether current curriculum guidelines provide the necessary guidance for teaching adversarial thinking in the classroom. The chapter answers the research question:

> **RQ3**: Can Sternberg's triarchic theory of intelligence provide a paradigm for defining adversarial thinking for cybersecurity that identifies practicable student learning outcomes?

It demonstrates how Sternberg's theory provides a helpful lens for unpacking what it really means to "think like a hacker," and this exercise produces a novel definition that sheds new light on the characteristic thought processes of proficient hackers. Most beneficially, the new definition leads directly to three new and well-defined learning outcomes for cybersecurity, including one that draws attention to the importance of strategic reasoning for adversarial thinking. Furthermore, the chapter suggests that strategic reasoning is a skill which has the potential to be developed in cybersecurity students.

Chapter 4 takes up the challenge of developing the strategic reasoning abilities of cybersecurity students. It answers the research question:

> **RQ4**: Does learning basic game theory concepts improve a student's ability to anticipate the strategic choices made by other people?

With a pretest-posttest educational experiment that includes a control group and an original measurement instrument, it demonstrates that learning basic game theory concepts results in a statistically significant improvement in a student's ability to anticipate the strategic choices made by others. Additionally, the chapter provides curriculum details which will aid other cybersecurity educators who seek to teach basic game theory concepts in their classrooms. The chapter also presents evidence gathered from student oral interviews that suggests that learning about game theory in a cybersecurity course has the potential to help to orient students around the adversarial conflict at the heart of cybersecurity. The research findings in this chapter could lead to a more strategic-minded, and therefore better equipped, cybersecurity workforce

And finally, Chapter 5 tackles the problem of rigorously characterizing the fault tolerance of CPSs. Critically assessing the reliability of such systems is non-trivial due to their inherent complexity and concurrency which makes reasoning about their operation, especially in light of various combinations of failure scenarios, difficult. The chapter answers the research question:

> **RQ5**: Can the SPIN model checker be applied to automate the identification of the degree of fault tolerance of CPSs?

It demonstrates that SPIN can be applied in an iterative manner to determine the degree of fault tolerance of a published decentralized peer-to-peer CPS, for which only anecdotal evidence of its robustness based on four failure scenario simulations exists. Over 275,000 failure scenarios were examined during the SPIN tests that were performed on the system's core decision algorithm, and these tests prove in a brute-force manner the precise degree of the system's fault tolerance. The same technique applied in this chapter

is applicable to a wide variety of CPS software designs, and it provides key insights into understanding the security vulnerabilities of such systems.

In summary, this dissertation has specifically made four main contributions:

- Integrates behavioral game theory concepts into an integer linear program to strategically allocate security resources. This program bested nearly all of the human competitors in an attack and defend competition, indicating high effectiveness against intelligent adversaries.
- Defines a new framework for adversarial thinking for cybersecurity based on Sternberg's triarchic theory of intelligence. The new definition provides actionable student learning outcomes, which was a weakness of previous definitions.
- Demonstrates the impact of teaching basic game theory concepts to cybersecurity students through a pretest-posttest educational experiment.
- Illustrates the power of model checking to precisely identify the degree of fault-tolerance for smart grid protection systems, which are an important category of CPSs. The applied model checking technique is applicable to a wide variety of CPS software designs, and it provides key insights into understanding the security vulnerabilities of CPSs.

6.1 Future Work

There are at least two future research directions that are natural continuations of this dissertation. The first could attempt to apply the research findings from Chapter 5 to a cybersecurity educational context, similarly to the way the game theory research from Chapter 1 was successfully applied to cybersecurity education in Chapters 2 and 3. This research could examine the question:

R6: Does learning how to use SPIN improve a student's ability to write more secure and reliable software?

Specifically, this research would emphasize how model checking, which is a software engineering best practice in safety-critical industries such as the aerospace and aeronautical industries, is not currently being included in secure software development modules in cybersecurity educational curriculums. While students are educated on many different types of software testing, including security testing, these forms of software testing are inadequate for verifying distributed software—the increasingly dominant type of software being produced today due to the proliferation of network-based applications. It is notoriously difficult to write error-free distributed software because of its inherent concurrency, which leads to a multitude of possible ways that program states and communications can interleave. In today's world of cyber warfare, software bugs become exploits, and when it comes CPSs like the smart grid, exploits become threats to society. Therefore, researching methods to improve the next generation of software developers' abilities to write more secure and reliable software could make a significant contribution to the future of cybersecurity.

The second research direction that is a natural continuation of this dissertation would be to combine the research findings from Chapter 2 on behavioral game theory, with the findings from Chapter 5 on SPIN, into a novel, two-phased approach to improving CPS cybersecurity. The top-down approach of Chapter 2 and the bottom-up approach of Chapter 5 have the potential to be unified, creating a novel technique capable of identifying the most vulnerable points in a CPS (the SPIN research) and then reinforcing precisely those points in the most efficient manner possible (the behavioral

game theory research). This two-phased approach could potentially inform a highly strategic protection posture which would provide a meaningful contribution to the cybersecurity of CPS.

Appendix A: The Data Breach Measurement Instrument

Data Breach Exercise - Defense

Imagine a large company with a deeply entrenched and ancient mainframe computer where they collect new customer data. The mainframe is difficult to secure due to technology constraints. To mitigate the damage from a data breach, every weekend they run a large job that moves all of the data off of the mainframe and onto a more secure server.

During any given week they are concerned that an insider might copy all of the customer data off of the mainframe and sell it on the black market. The only deterrent they have against such an attack is the threat of auditing the log files, and going forward they have decided to allocate 100 man hours per week to that task.

They collect about the same amount of data each day, therefore, the database grows linearly throughout the week. The database starts fresh on Monday mornings because of the weekend migration job. For simplicity, assume that the number of hours allocated to inspecting the logs equals the likelihood of detecting an attack. For example, if *x* hours are assigned to a particular day's logs, and an insider attacks on that day, then the chance of detecting the insider is *x* percent. Also assume that if the insider is detected, the threat will be eliminated resulting in a "reward" equal to 10 points for the company.

They have hired you as a cybersecurity consultant because they need help. Your job is to allocate the 100 man hours over the 5 log files. Fill in the table below with *integers* in the range [0, 100] and make sure they sum to 100.

Log files:	Monday	Tuesday	Wednesday	Thursday	Friday
Value of database:	**1**	**2**	**3**	**4**	**5**
Hours spent auditing logs: **(must sum to 100)**					

The company that hired you wants to know how you came up with this particular allocation of hours. *Briefly* describe what you would tell them:

Appendix B: The SPIN Model of Tong's WABPS

```
// this is like a struct in C++, and encapsulates an IED's state
 typedef ied {
   byte id;   // ied number
   mtype dir; // directional_relay
   mtype pri; // primary_relay
   mtype sec; // secondary_relay
   bool dirOn;
   bool priOn;
   bool secOn;
};

// globally defined IEDs (there are 24 of them)
ied IED29;
ied IED30;
ied IED15;
ied IED16;
ied IED21;
ied IED22;
ied IED27;
ied IED28;
ied IED03;
ied IED04;
ied IED05;
ied IED06;
ied IED07;
ied IED08;
ied IED11;
ied IED12;
ied IED13;
ied IED14;
ied IED17;
ied IED18;
ied IED19;
ied IED20;
ied IED23;
ied IED24;

// these are the possible line states
mtype = { NO_DATA, NORMAL, SPECIAL, SUSPECT, FAULTED };
// these are the possible directional relay states
mtype = { LINE_FAULT, BUS_FAULT, NONE };
// these are the possible primary and secondary relay states
mtype = { FAULT, NO_FAULT };

// these are the arrays that hold the verdicts for the 4 line states for each of the 4 LDAs
mtype LDA15Verdict[4] = { NO_DATA, NO_DATA, NO_DATA, NO_DATA };
mtype LDA12Verdict[4] = { NO_DATA, NO_DATA, NO_DATA, NO_DATA };
mtype LDA14Verdict[4] = { NO_DATA, NO_DATA, NO_DATA, NO_DATA };
mtype LDA09Verdict[4] = { NO_DATA, NO_DATA, NO_DATA, NO_DATA };

// this is an array of 4 channels, 1 for each line L9, L12, L14, L15, each capable of holding 4 msgs
// the message itself will be a 4-tuple containing:
// line number, line state, Fout value, and number of protection actions
chan LDAChan[4] = [4] of { byte, mtype, short, byte };

// these bools help SPIN run faster by dictating the order the LDAs fire
bool line15Started = false;
bool line12Started = false;
bool line14Started = false;

// before and after the init function, there are lots of macros. PROMELA doesn't support functions
// so this "hack" makes the code somewhat modular and maintainable, although macros aren't nearly as
// easy to read as functions, so the code is more complex than I would have liked

// this is a utility for setting the state variables on an IED
#define initIED(iedXX, num, dirState, priState, secState) \
```

```
67    iedXX.id = num; \
68    iedXX.dir = dirState; \
69    iedXX.pri = priState; \
70    iedXX.sec = secState
71
72   // these are the normal states of all 24 IEDS when a fault occurs on L15
73   #define initIEDStatesForL15Fault() \
74    initIED(IED29, 29, LINE_FAULT, FAULT, NO_FAULT); \
75    initIED(IED30, 30, LINE_FAULT, FAULT, NO_FAULT); \
76    initIED(IED28, 28, BUS_FAULT, NO_FAULT, NO_FAULT); \
77    initIED(IED27, 27, LINE_FAULT, NO_FAULT, NO_FAULT); \
78    initIED(IED22, 22, BUS_FAULT, NO_FAULT, NO_FAULT); \
79    initIED(IED21, 21, LINE_FAULT, NO_FAULT, NO_FAULT); \
80    initIED(IED16, 16, BUS_FAULT, NO_FAULT, NO_FAULT); \
81    initIED(IED15, 15, LINE_FAULT, NO_FAULT, NO_FAULT); \
82    initIED(IED14, 14, BUS_FAULT, NO_FAULT, NO_FAULT); \
83    initIED(IED13, 13, LINE_FAULT, NO_FAULT, NO_FAULT); \
84    initIED(IED23, 23, LINE_FAULT, NO_FAULT, NO_FAULT); \
85    initIED(IED24, 24, BUS_FAULT, NO_FAULT, NO_FAULT); \
86    initIED(IED04, 4, BUS_FAULT, NO_FAULT, NO_FAULT); \
87    initIED(IED03, 3, LINE_FAULT, NO_FAULT, NO_FAULT); \
88    initIED(IED08, 8, BUS_FAULT, NO_FAULT, NO_FAULT); \
89    initIED(IED07, 7, LINE_FAULT, NO_FAULT, NO_FAULT); \
90    initIED(IED06, 6, BUS_FAULT, NO_FAULT, NO_FAULT); \
91    initIED(IED05, 5, LINE_FAULT, NO_FAULT, NO_FAULT); \
92    initIED(IED12, 12, BUS_FAULT, NO_FAULT, NO_FAULT); \
93    initIED(IED11, 11, LINE_FAULT, NO_FAULT, NO_FAULT); \
94    initIED(IED17, 17, BUS_FAULT, NO_FAULT, NO_FAULT); \
95    initIED(IED18, 18, LINE_FAULT, NO_FAULT, NO_FAULT); \
96    initIED(IED19, 19, LINE_FAULT, NO_FAULT, NO_FAULT); \
97    initIED(IED20, 20, BUS_FAULT, NO_FAULT, NO_FAULT)
98
99   // these are the 4 failure scenarios from Tong's WABPS research paper
100  // in simulation mode, one scenario will be selected nondeterministically
101  // in verification mode, they will all be tested
102  #define papersTestScenarios() \
103   if \
104   ::printf("Scenario A test\n"); \
105    initIED(IED29, 29, NONE, NO_FAULT, NO_FAULT); \
106    initIED(IED30, 30, NONE, NO_FAULT, NO_FAULT); \
107   ::printf("Scenario A test\n"); \
108    initIED(IED28, 28, LINE_FAULT, NO_FAULT, FAULT); \
109    initIED(IED16, 16, BUS_FAULT, NO_FAULT, FAULT); \
110   ::printf("Scenario C test\n"); \
111    initIED(IED30, 30, NONE, FAULT, NO_FAULT); \
112    initIED(IED22, 22, NONE, NO_FAULT, NO_FAULT); \
113   ::printf("Scenario D test\n"); \
114    initIED(IED29, 29, BUS_FAULT, FAULT, NO_FAULT); \
115    initIED(IED30, 30, LINE_FAULT, NO_FAULT, NO_FAULT); \
116    initIED(IED28, 28, LINE_FAULT, NO_FAULT, NO_FAULT); \
117    initIED(IED22, 22, LINE_FAULT, NO_FAULT, NO_FAULT); \
118   fi
119
120  // this is the test for when any given component of an IED can fail
121  #define corruptIEDComponent(iedXX) \
122   if \
123   ::(!iedXX.dirOn)->iedXX.dir = LINE_FAULT; iedXX.dirOn=true; printf("Corrupted Component IED%d, dir
= LINE_FAULT\n", iedXX.id); \
124   ::(!iedXX.dirOn)->iedXX.dir = BUS_FAULT;  iedXX.dirOn=true; printf("Corrupted Component IED%d, dir
= BUS_FAULT\n", iedXX.id); \
125   ::(!iedXX.dirOn)->iedXX.dir = NONE;       iedXX.dirOn=true; printf("Corrupted Component IED%d, dir
= NONE\n", iedXX.id); \
126   ::(!iedXX.priOn)->iedXX.pri = FAULT;      iedXX.priOn=true; printf("Corrupted Component IED%d, pri
= FAULT\n", iedXX.id); \
127   ::(!iedXX.priOn)->iedXX.pri = NO_FAULT;   iedXX.priOn=true; printf("Corrupted Component IED%d, pri
= NO_FAULT\n", iedXX.id); \
128   ::(!iedXX.secOn)->iedXX.sec = FAULT;      iedXX.secOn=true; printf("Corrupted Component IED%d, sec
= FAULT\n", iedXX.id); \
```

```
 ::(!iedXX.secOn)->iedXX.sec = NO_FAULT;   iedXX.secOn=true; printf("Corrupted Component IED%d, sec
= NO_FAULT\n", iedXX.id); \
 fi

// this is the test when an entire IED can fail
#define turnOffIED(iedXX) \
 printf("Turned Off IED%d\n", iedXX.id); \
 iedXX.dir = NONE; iedXX.dirOn=true; \
 iedXX.pri = NO_FAULT; iedXX.priOn=true; \
 iedXX.sec = NO_FAULT; iedXX.secOn=true;

// this checks that the IED can be selected by the tests by making sure it hasn't been
// totally corrupted yet
#define iedIsSelectable(iedXX) \
 !(iedXX.dirOn && iedXX.priOn && iedXX.secOn)

// test that at least 1 of the LDAs identified the fault on L15 (weak correctness)
#define verifyL15Fault() \
 assert (LDA15Verdict[0] == FAULTED || LDA12Verdict[0] == FAULTED || LDA14Verdict[0] == FAULTED ||
LDA09Verdict[0] == FAULTED)

// test that no LDA determined any other line was faulted (strong correctness)
#define verifyOnlyL15Fault() \
 assert (LDA15Verdict[1] != FAULTED && LDA15Verdict[2] != FAULTED && LDA15Verdict[3] != FAULTED); \
 assert (LDA12Verdict[1] != FAULTED && LDA12Verdict[2] != FAULTED && LDA12Verdict[3] != FAULTED); \
 assert (LDA14Verdict[1] != FAULTED && LDA14Verdict[2] != FAULTED && LDA14Verdict[3] != FAULTED); \
 assert (LDA09Verdict[1] != FAULTED && LDA09Verdict[2] != FAULTED && LDA09Verdict[3] != FAULTED)

// this is the generic test method
// the first arg is a "function pointer" to the specific failure to be tested
// the second arg is the maximum number of failures to be tested
#define test(failureType, maxFailures) \
failuresSoFar = 0; \
 do \
 ::(failuresSoFar <  maxFailures) -> \
  if \
  ::(iedIsSelectable(IED29))->failureType(IED29); \
  ::(iedIsSelectable(IED30))->failureType(IED30); \
  ::(iedIsSelectable(IED28))->failureType(IED28); \
  ::(iedIsSelectable(IED27))->failureType(IED27); \
  ::(iedIsSelectable(IED22))->failureType(IED22); \
  ::(iedIsSelectable(IED21))->failureType(IED21); \
  ::(iedIsSelectable(IED16))->failureType(IED16); \
  ::(iedIsSelectable(IED15))->failureType(IED15); \
  ::(iedIsSelectable(IED14))->failureType(IED14); \
  ::(iedIsSelectable(IED13))->failureType(IED13); \
  ::(iedIsSelectable(IED23))->failureType(IED23); \
  ::(iedIsSelectable(IED24))->failureType(IED24); \
  ::(iedIsSelectable(IED04))->failureType(IED04); \
  ::(iedIsSelectable(IED03))->failureType(IED03); \
  ::(iedIsSelectable(IED08))->failureType(IED08); \
  ::(iedIsSelectable(IED07))->failureType(IED07); \
  ::(iedIsSelectable(IED06))->failureType(IED06); \
  ::(iedIsSelectable(IED05))->failureType(IED05); \
  ::(iedIsSelectable(IED12))->failureType(IED12); \
  ::(iedIsSelectable(IED11))->failureType(IED11); \
  ::(iedIsSelectable(IED17))->failureType(IED17); \
  ::(iedIsSelectable(IED18))->failureType(IED18); \
  ::(iedIsSelectable(IED19))->failureType(IED19); \
  ::(iedIsSelectable(IED20))->failureType(IED20); \
  fi; \
  failuresSoFar++; \
 ::else -> break; \
 od

init {
 // modify this to measure a particular fault tolerance degree
 byte numSingleComponentFailuresToTest = 1;
 byte numTotalIEDFailuresToTest = 1;
```

```
196  byte failuresSoFar = 0
197
198  initIEDStatesForL15Fault();
199
200  // these are a list of tests that may be executed, some will be commented out
201  //testPapersFailureScenarios();
202  //test(corruptIEDComponent, numSingleComponentFailuresToTest);
203  test(turnOffIED, numTotalIEDFailuresToTest);
204
205  // this is atomic so the verification mode will run faster
206  // arbirary interleavings do not change the results, so I am not
207  // making SPIN perform them all
208  atomic {
209    run LDA15(0, 15);
210    run LDA12(1, 12);
211    run LDA14(2, 14);
212    run LDA09(3, 9);
213  }
214
215  // this blocks the init process until all the LDAs have completed
216  do
217  ::timeout -> break
218  od;
219
220  verifyL15Fault();  // weak correctness test
221  verifyOnlyL15Fault(); // strong correctness test
222
223 }
224
225 // the formula is at the bottom of page 1198 in Tong's paper,
226 // but this is the C code which shows how it was meant to be implemented:
227 // if (directional == 1) {
228 //   if (primary_relay == 1) {
229 //   AF = 1;
230 //  } else if (second_relay == 1) {
231 //   AF = 0.5;
232 //  }
233 // } else if ( (directional == -1) && (primary_relay == 0) && (second_relay == 0)) {
234 //  AF = -1;
235 // } else {
236 //  AF = 0;
237 // }
238 // the values had to be mapped to ints since SPIN doesn't support floating point numbers:
239 //  1 -> 2
240 // .5 -> 1
241 // -1 -> -2
242 //  0 -> 0
243 #define calcAF(iedXX, af) \
244  if \
245  ::((iedXX.dir == LINE_FAULT) && (iedXX.pri == FAULT)) -> af = 2; \
246  ::((iedXX.dir == LINE_FAULT) && (iedXX.sec == FAULT) && (iedXX.pri == NO_FAULT)) -> af = 1; \
247  ::((iedXX.dir == BUS_FAULT) && (iedXX.pri == NO_FAULT) && (iedXX.sec == NO_FAULT)) -> af = -2; \
248  ::else -> af = 0; \
249  fi
250
251 #define calcNumberOfProtectionActions(iedXX) \
252  if \
253  ::(iedXX.dir != NONE) -> lineProtectionActions++; \
254  ::else -> skip; \
255  fi; \
256  if \
257  ::(iedXX.pri == FAULT) -> lineProtectionActions++; \
258  ::else -> skip; \
259  fi; \
260  if \
261  ::(iedXX.sec == FAULT) -> lineProtectionActions++; \
262  ::else -> skip; \
263  fi
264
```

```
// this is taken from the bottom of the right column on page 1198
#define Fset 2
#define calcLineStateFromFout() \
 if \
 ::(Fout < 0) -> lineState = NORMAL; \
 ::((Fout >= 0) && (Fout < Fset)) -> lineState = SPECIAL; \
 ::(Fout == Fset) -> lineState = SUSPECT; \
 ::else -> lineState = FAULTED; \
 fi

// STEP 1: calculate the AF value, which is based on the state of the IEDs on my line
#define doStage1(iedXX, iedYY) \
 calcAF(iedXX, af1); \
 calcAF(iedYY, af2); \
 Fout = af1 + af2; \
 lineProtectionActions = 0; \
 calcNumberOfProtectionActions(iedXX); \
 calcNumberOfProtectionActions(iedYY); \
 calcLineStateFromFout(); \
 printf("Step 1 - L%d State: %e\n", actualLine, lineState);

// do not have type double, but the rule is if at least half of the indicators
// indicate a fault, then the threshold is exceeded
#define calcCF(sharedBus, nonSharedBus, cnt, cfThresholdExceeded) \
 sum = 0; \
 sbIndex = 0; \
 do \
 ::(sbIndex < cnt) -> \
  if \
  ::(sharedBus[sbIndex].dir == BUS_FAULT) -> sum++; \
  ::else -> skip; \
  fi; \
  sbIndex++; \
 ::(sbIndex == cnt) -> break; \
 od; \
 nsbIndex = 0; \
 do \
 ::(nsbIndex < cnt) -> \
  if \
  ::(nonSharedBus[nsbIndex].dir == LINE_FAULT) -> sum++; \
  ::else -> skip; \
  fi; \
  nsbIndex++; \
 ::(nsbIndex == cnt) -> break; \
 od; \
 cfThresholdExceeded = (( sum*2) >= cnt )

// both have to be above the threshold to move from SPECIAL to SUSPECT
#define calcLineStateFromCFs() \
 if \
 ::( cf1ThresholdExceeded && cf2ThresholdExceeded ) -> lineState = SUSPECT; \
 ::else -> lineState = NORMAL; \
 fi

// STEP 2: if we are in SPECIAL state, either escalate to SUSPECT or de-escalate to NORMAL
// which is based on CF values, which are calculated the IEDs on neighboring lines
#define doStage2() \
 if \
 ::(lineState == SPECIAL) -> \
  calcCF(cf1SB, cf1NSB, cf1Cnt, cf1ThresholdExceeded); \
  calcCF(cf2SB, cf2NSB, cf2Cnt, cf2ThresholdExceeded); \
  calcLineStateFromCFs(); \
 ::else -> skip; \
 fi; \
 printf("Step 2 - L%d State: %e\n", actualLine, lineState)

// STEP 3: if we are in SUSPECT state, either escalate to FAULT or de-escalate to NORMAL
// However, if we de-escalate to NORMAL, then we must determine another line to have a FAULT
// this is based on the fault state of neighboring lines
```

```
#define doStage3(verdict) \
 sendStateToNeighbors(); \
 if \
 ::(lineState == SUSPECT) -> \
  receiveStateFromNeighbors(verdict); \
 ::else -> skip; \
 fi; \
 if \
 ::(lineState == SUSPECT) -> \
  resolveSuspectState(verdict); \
 ::else -> skip; \
 fi; \
 printf("Step 3 - L%d State: %e\n", actualLine, lineState); \
 verdict[lineNumber] = lineState

#define sendStateToNeighbors() \
 sendLineNum = 0; \
 do \
 ::(sendLineNum < 4) -> \
  if \
  ::(sendLineNum != lineNumber) -> LDAChan[sendLineNum]!lineNumber(lineState, Fout,
lineProtectionActions); \
  ::else -> skip; \
  fi; \
  sendLineNum++; \
 ::(sendLineNum == 4) -> break; \
 od

#define receiveStateFromNeighbors(verdict) \
 msgsRcvd = 0; \
 maxFout = Fout; \
 maxProtectionActions = 0; \
 maxFoutLineNumber = lineNumber; \
 do \
 ::(msgsRcvd < 3) -> LDAChan[lineNumber]?neighborLineNumber(neighborLineState, neighborFout,
neighborProtectionActions); \
  msgsRcvd++; \
  if \
  ::(neighborLineState == FAULTED) -> \
   lineState = NORMAL; \
   verdict[neighborLineNumber] = FAULTED; \
   break; \
  ::(neighborLineState == NORMAL) -> skip; \
  ::(neighborLineState == SUSPECT) -> \
   if \
   ::((neighborFout > maxFout) ||  \
    ((neighborFout == maxFout) && (neighborProtectionActions > lineProtectionActions))) -> \
    maxFout = neighborFout; \
    maxFoutLineNumber = neighborLineNumber; \
    maxProtectionActions = neighborProtectionActions; \
   ::else -> skip; \
   fi; \
  ::else -> skip; \
  fi; \
 ::(msgsRcvd == 3) -> break; \
 od; \

#define resolveSuspectState(verdict) \
 if \
 ::(maxFout > Fout) -> \
  lineState = NORMAL; \
  verdict[maxFoutLineNumber] = FAULTED; \
 ::(maxFout == Fout) -> \
  if \
  ::(maxProtectionActions > lineProtectionActions) -> \
   lineState = NORMAL; \
   verdict[maxFoutLineNumber] = FAULTED; \
  ::(maxProtectionActions == lineProtectionActions) -> \
   lineState = FAULTED; \
```

```
401     verdict[maxFoutLineNumber] = FAULTED; \
402    ::else -> \
403     lineState = FAULTED; \
404     verdict[maxFoutLineNumber] = NORMAL; \
405    fi; \
406   ::else -> lineState = FAULTED; \
407   fi
408
409 #define initVars() \
410   mtype lineState = NO_DATA; \
411   short af1; \
412   short af2; \
413   short Fout; \
414   byte lineProtectionActions; \
415   byte sbIndex; \
416   byte nsbIndex; \
417   bool cf1ThresholdExceeded; \
418   bool cf2ThresholdExceeded; \
419   byte sum; \
420   byte sendLineNum; \
421   byte msgsRcvd; \
422   mtype neighborLineState; \
423   short neighborFout; \
424   byte neighborLineNumber; \
425   byte neighborProtectionActions; \
426   short maxFout; \
427   byte maxFoutLineNumber; \
428   byte maxProtectionActions
429
430 // SB stands for Shared Bus, NSB stands for Non Shared Bus
431 #define initIEDArrays(cnt1, cnt2)  \
432   byte cf1Cnt = cnt1; \
433   byte cf2Cnt = cnt2; \
434   ied cf1SB[cnt1]; \
435   ied cf1NSB[cnt1]; \
436   ied cf2SB[cnt2]; \
437   ied cf2NSB[cnt2]
438
439 #define initArray(arr, iedXX, i) \
440   arr[i].id  = iedXX.id; \
441   arr[i].dir = iedXX.dir; \
442   arr[i].pri = iedXX.pri; \
443   arr[i].sec = iedXX.sec
444
445 #define runAlgorithm(iedXX, iedYY, verdict) \
446   printf("LDA%d Start\n", actualLine); \
447   doStage1(iedXX, iedYY); \
448   doStage2(); \
449   doStage3(verdict); \
450   printf("LDA%d Final: L15:%e, L12:%e, L14:%e, L9:%e\n", actualLine, verdict[0], verdict[1],
verdict[2], verdict[3])
451
452 // NAME    PRIMARY    CF1_SHARED   CF1_NONSHARED   CF2_SHARED   CF2_NONSHARED
453 // LDA15   (29, 30)   (22, 28)     (21, 27)        (16)         (15)
454 proctype LDA15(byte lineNumber, actualLine) {
455   atomic {
456     line15Started = true;
457     initVars();
458     initIEDArrays(2, 1);
459     initArray(cf1SB, IED22, 0);
460     initArray(cf1SB, IED28, 1);
461     initArray(cf1NSB, IED21, 0);
462     initArray(cf1NSB, IED27, 1);
463     initArray(cf2SB, IED16, 0);
464     initArray(cf2NSB, IED15, 0);
465     runAlgorithm(IED29, IED30, LDA15Verdict);
466   }
467 }
468
```

```
// NAME   PRIMARY   CF1_SHARED  CF1_NONSHARED  CF2_SHARED          CF2_NONSHARED
// LDA12  (21, 22)  (28, 29)    (27, 30)       (4, 6, 14, 19, 23)  (3, 5, 13, 20, 24)
proctype LDA12(byte lineNumber, actualLine) {
 atomic {
   line15Started->skip;
   line12Started = true;
   initVars();
   initIEDArrays(2, 5);
   initArray(cf1SB, IED28, 0);
   initArray(cf1SB, IED29, 1);
   initArray(cf1NSB, IED27, 0);
   initArray(cf1NSB, IED30, 1);
   initArray(cf2SB, IED04, 0);
   initArray(cf2SB, IED06, 1);
   initArray(cf2SB, IED14, 2);
   initArray(cf2SB, IED19, 3);
   initArray(cf2SB, IED23, 4);
   initArray(cf2NSB, IED03, 0);
   initArray(cf2NSB, IED05, 1);
   initArray(cf2NSB, IED13, 2);
   initArray(cf2NSB, IED20, 3);
   initArray(cf2NSB, IED24, 4);
   runAlgorithm(IED21, IED22, LDA12Verdict);
 }
}

// NAME   PRIMARY   CF1 SHARED  CF1 NONSHARED  CF2 SHARED  CF2 NONSHARED
// LDA14  (27, 28)  (22, 29)    (21, 30)       (24)        (23)
proctype LDA14(byte lineNumber, actualLine) {
 atomic {
   line12Started->skip;
   line14Started = true;
   initVars();
   initIEDArrays(2, 1);
   initArray(cf1SB, IED22, 0);
   initArray(cf1SB, IED29, 1);
   initArray(cf1NSB, IED21, 0);
   initArray(cf1NSB, IED30, 1);
   initArray(cf2SB, IED24, 0);
   initArray(cf2NSB, IED23, 0);
   runAlgorithm(IED27, IED28, LDA14Verdict);
 }
}

// NAME   PRIMARY   CF1 SHARED  CF1 NONSHARED  CF2 SHARED        CF2 NONSHARED
// LDA9   (15, 16)  (30)        (29)           (8, 12, 13, 17)   (7, 11, 14, 18)
proctype LDA09(byte lineNumber, actualLine) {
 atomic {
   line14Started->skip;
   initVars();
   initIEDArrays(1, 4);
   initArray(cf1SB, IED30, 0);
   initArray(cf1NSB, IED29, 0);
   initArray(cf2SB, IED08, 0);
   initArray(cf2SB, IED12, 1);
   initArray(cf2SB, IED13, 2);
   initArray(cf2SB, IED17, 3);
   initArray(cf2NSB, IED07, 0);
   initArray(cf2NSB, IED11, 1);
   initArray(cf2NSB, IED14, 2);
   initArray(cf2NSB, IED18, 3);
   runAlgorithm(IED15, IED16, LDA09Verdict);
 }
}

```

Bibliography

[1] E. A. Lee, "Cyber physical systems: design challenges," in *2008 11th IEEE international symposium on object and component-oriented real-time distributed computing*, Orlando, 2008.

[2] P. McDaniel and S. McLaughlin, "Security and privacy challenges in the smart grid," *IEEE Security and Privacy,* vol. 7, no. 3, pp. 75-77, 2009.

[3] N. Jazdi, "Cyber physical systems in the context of Industry 4.0," in *2014 IEEE international conference on automation, quality and testing, robotics*, Cluj-Napoca, 2014.

[4] K. Zhou, T. Liu and L. Zhou, "Industry 4.0: towards future industrial opportunities and challenges," in *Twelfth international conference on fuzzy systems and knowledge discovery*, Zhangjiajie, 2015.

[5] G. Brown, M. Carlyle, J. Salmeron and K. Wood, "Defending critical infrastructure," *Interfaces,* vol. 36, no. 6, pp. 530-544, 2006.

[6] C. Neuman, "Challenges in security for cyber-physical systems," DHS workshop on future directions in cyber-physical systems security, Newark, 2009.

[7] K. K. Fletcher and X. Liu, "Security requirements analysis, specification, prioritization, and policy development in cyber-physical systems," in *Fifth international conference on secure software integration and reliability improvement companion*, Jeju Island, 2011.

[8] U.S. Office of Homeland Security, "National strategy for homeland security," Office of Homeland Security, Washington, D.C., 2002.

[9] A. Fielder, E. Panaousis, P. Malacaria, C. Hankin and F. Smeraldi, "Game theory meets information security management," in *Proceedings of the twenty-ninth IFIP TC 11 international conference*, Marrakech, 2014.

[10] X. Feng, T. Lu, X. Guo, J. Liu, Y. Peng and Y. Gao, "Security analysis on cyber-physical systems using attack tree," in *Ninth international conference on intelligent information hiding and multimedia signal processing*, Beijing, 2013.

[11] A. Cardenas, S. Amin, B. Sinopoli, A. Giani, A. Perrig and S. Sastry, "Challenges for securing cyber physical systems," Workshop on future directions in cyber-physical systems security, Newark, 2009.

[12] F. He, J. Zhuang and N. Rao, "Game-theoretic analysis of attack and defense in cyber-physical network infrastructures," in *Proceedings of the industrial and systems engineering research conference*, Orlando, 2012.

[13] C. Y. Ma, N. S. Rao and D. K. Yau, "A game theoretic study of attack and defense in cyber-physical systems," in *IEEE conference on computer communications workshops*, Shanghai, 2011.

[14] R. Vigo, A. Bruni and E. Yuksel, "Security games for cyber-physical systems," in *Proceedings of the eighteenth Nordic conference*, Oslo, 2013.

[15] S. Backhaus, R. Bent, J. Bono, R. Lee, B. Tracey, D. Wolpert, D. Xie and Y. Yildiz, "Cyber-physical security: a game theory model of humans interacting over control systems," *IEEE Transactions on Smart Grid*, vol. 4, no. 4, pp. 2320-2327, 2013.

[16] M. Tambe, Security and game theory: algorithms, deployed systems, lessons learned, New York: Cambridge University Press, 2012.

[17] T. Alpcan and T. Basar, A decision and game-theoretic approach, United Kingdom: Cambridge University Press, 2011.

[18] C. Camerer, Behavioral game theory, Princeton: Princeton University Press, 2003.

[19] A. Arad and A. Rubinstein, "The 11-20 money request game: a level-k reasoning study," *American Economic Review*, vol. 102, no. 7, pp. 3561-3573, 2012.

[20] O. Gross and R. Wagner, "A continuous Colonel Blotto game," RAND Corporation, 1950.

[21] A. Arad and A. Rubinstein, "Multi-dimensional iterative reasoning in action: the case of the Colonel Blotto game," *Journal of Economic Behavior & Organization*, vol. 84, no. 2, pp. 571-585, 2012.

[22] A. Arad and A. Rubinstein, "Let game theory be?," *Calcalist,* 11 May 2009.

[23] Lloyds, "Business blackout: the insurance implications of a cyber attack on the U.S. power grid," Lloyds, London, 2015.

[24] U.S. Department of Energy, "Final report on the August 14, 2003 blackout in the United States and Canada: causes and recommendations," U.S. Department of Energy, Washington, D.C., 2004.

[25] K. J. Ross, K. M. Hopkinson and M. Pachter, "Using a distributed agent-based communication enabled special protection system to enhance smart grid security," *IEEE Transactions on Smart Grid,* vol. 4, no. 2, pp. 1216-1224, 2013.

[26] F. B. Schneider, "Cybersecurity education in universities," *IEEE Security & Privacy,* vol. 11, no. 4, pp. 3-4, 2013.

[27] A. McGettick, "Toward curricular guidelines for cybersecurity," 30 August 2013. [Online]. Available: http://www.acm.org/education/TowardCurricularGuidelinesCybersec.pdf. [Accessed 15 March 2016].

[28] The Joint Task Force on Computing Curricula, "Computer science curricula 2013," 20 December 2013. [Online]. Available: http://www.acm.org/education/CS2013-final-report.pdf. [Accessed 15 March 2016].

[29] The National Security Agency, "Academic requirements for designation as a center of academic excellence in cyber operations," 9 November 2015. [Online]. Available: https://www.nsa.gov/academia/nat_cae_cyber_ops/nat_cae_co_requirements.shtml. [Accessed 15 March 2016].

[30] M. Dark and J. Mirkovic, "Evaluation theory and practice applied to cybersecurity education," *IEEE Security & Privacy,* vol. 13, no. 2, pp. 75-80, 2015.

[31] American Psychology Association, "Glossary of psychological terms," American Psychology Association, [Online]. Available: http://www.apa.org/research/action/glossary.aspx?tab=3. [Accessed 15 March 2016].

[32] R. Sternberg, The triarchic mind, New York: Penguin Books, 1988.

[33] R. B. Tigner and S. S. Tigner, "Triarchic theories of intelligence: Aristotle and Sternberg," *History of Psychology,* vol. 3, no. 2, pp. 168-176, 2000.

[34] R. T. Morris, "A weakness in the 4.2 BSD Unix TCP/IP software," AT&T Bell Laboratories, Murray Hill, 1985.

[35] K. Hafner and J. Markoff, Cyberpunk, New York: Touchstone, 1995.

[36] E. Levy, "Smashing the stack for fun and profit," *Phrack,* vol. 7, no. 49, p. 14, 1996.

[37] B. Schneier, "What is a hacker?," 14 September 2006. [Online]. Available: https://www.schneier.com/blog/archives/2006/09/what_is_a_hacke.html. [Accessed 15 March 2016].

[38] Wikipedia contributors, "IP fragmentation attack," Wikipedia, 13 January 2016. [Online]. Available: https://en.wikipedia.org/wiki/IP_fragmentation_attack. [Accessed 15 March 2016].

[39] E. Skoudis, Counter hack reloaded, Boston: Pearson, 2006.

[40] T. C. Summers, "How hackers think: a study of cybersecurity experts and their mental models," in *Third Annual International Conference on Engaged Management Scholarship,* Atlanta, 2013.

[41] K. Mitnick and W. L. Simon, Ghost in the wires, New York: Back Bay Books, 2012.

[42] C. Stoll, "Stalking the wily hacker," *Communications of the ACM,* vol. 31, no. 5, pp. 484-497, 1988.

[43] C. Stoll, The cuckoo's egg, New York: Pocket Books, 1990.

[44] G. Conti and J. Caroland, "Embracing the Kobayashi Maru: why you should teach your students to cheat," *IEEE Security & Privacy,* vol. 9, no. 4, pp. 48-51, 2011.

[45] B. Mullins, "Developing cyber warriors from computer engineers et al.," in *2012*

ASEE Annual Conference, San Antonio, TX, 2012.

[46] J. Mirkovic and P. A. H. Peterson, "Class capture-the-flag exercises," in *Proceedings of the USENIX Summit on Gaming, Games and Gamification in Security Education*, San Diego, 2014.

[47] J. Boleng, D. Schweitzer and D. S. Gibson, "Developing cyber warriors," in *The 3rd International Conference on Information Warfare and Security*, Omaha, 2008.

[48] Q. Campbell and D. M. Kennedy, "The psychology of computer criminals," in *Computer security handbook*, Hoboken, John Wiley & Sons, 2014, pp. 12:1-12:33.

[49] A. M. Colman, Game theory and its applications, New York: Routledge, 2003.

[50] J. Watson, Strategy: an introduction to game theory, New York: W. W. Norton & Company, 2013.

[51] A. M. Brandenburger and B. J. Nalebuff, Co-opetition, New York: Doubleday, 1996.

[52] R. Gash, "Game theory: can a round of poker solve Afghanistan's problems?," *Small Wars Journal*, pp. 1-4, 2009.

[53] Yale University, "MGT 525: competitive strategy," Yale University, [Online]. Available: http://faculty.som.yale.edu/FionaScottMorton/documents/syll_08.pdf. [Accessed 15 March 2016].

[54] Indiana University, "G570: thinking strategically: game theory and business strategy," Indiana University, [Online]. Available: http://kelley.iu.edu/BEPP/Masters/page14312.cfm?ID=9502. [Accessed 15 March 2016].

[55] J. R. Fraenkel and N. E. Wallen, How to design and evaluate research in education, New York: McGraw-Hill Higher Education, 2009.

[56] G. J. Holzmann, The SPIN model checker: primer and reference manual, Boston: Addison-Wesley, 2004.

[57] A. Khurram, H. Ali, A. Tariq and O. Hasan, "Formal reliability analysis of protective relays in power distribution systems," in *Proceedings of the 18th International Workshop on Formal Methods for Industrial Critical Systems*, Madrid, 2013.

[58] M. Moulin, L. Gluhovsky and D. Geist, "Formal verification analysis of load-voltage power dynamics and control," in *World Automation Congress 2004 Proceedings*, Seville, 2004.

[59] NASA, "Gerard Holzmann bio," NASA, 1 December 2009. [Online]. Available: http://lars-lab.jpl.nasa.gov/people/gh.html. [Accessed 18 May 2016].

[60] F. Wang and J. Tang, "Modeling of a transmission line protection relaying scheme using petri nets," *IEEE Transactions of Power Delivery,* vol. 12, no. 3, pp. 1055-1063, 1997.

[61] G. Ramos, J. L. Sanchez, A. Torres and M. A. Rios, "Power systems security evaluation using petri nets," *IEEE Transactions on Power Delivery,* vol. 25, no. 1, pp. 316-322, 2010.

[62] X. Tong, X. Wang and K. M. Hopkinson, "The modeling and verification of peer-to-peer negotiating multiagent colored petr nets for wide-area backup protection," *IEEE Transactions on Power Delivery,* vol. 24, no. 1, pp. 61-72, 2009.

[63] Z. Lin, F. Wen, C. Y. Chung and K. P. Wong, "A survey on the applications of petri net theory in power systems," in *IEEE Power Engineering Society General Meeting*, Montreal, 2006.

[64] T. Murata, "Petri nets: properties, analysis and applications," *Proceedings of the IEEE,* vol. 77, no. 4, pp. 541-580, 1989.

[65] A. Sengupta, S. Mukhopadhyay and A. K. Sinha, "Automated verification of power system protection schemes—part I: modeling and specifications," *IEEE Transactions on Power Delivery*, vol. 30, no. 5, pp. 2077-2086, 2015.

[66] T. A. Bopp, R. Ganjavi, R. Krebs, B. Ntsin, M. Dauer and J. Jaeger, "Improving grid reliability through application of protection security assessment," in *12th IET International Conference on Developments in Power System Protection*, Copenhagen, 2014.

[67] Official SPIN Website, "Inspiring applications of SPIN," Official SPIN Website, [Online]. Available: http://spinroot.com/spin/success.html. [Accessed 18 May 2016].

[68] F. Schneider, S. M. Easterbrook, J. R. Callahan and G. J. Holzmann, "Validating requirements for fault tolerant systems using model checking," in *Proceedings of Third International Conference on Requirements Engineering*, Colorado Springs, 1998.

[69] A. Gmeiner, I. Konnov, U. Schmid, H. Veith and J. Widder, "Tutorial on parameterized model checking of fault-tolerant distributed algorithms," in *Formal methods for executable software models*, Switzerland, Springer International Publishing, 2014, pp. 122-171.

[70] C. Newcombe, T. Rath, F. Zhang, B. Munteanu, M. Brooker and M. Deardeuff, "How Amazon web services uses formal methods," *Communications of the ACM*, pp. 66-73, March 2015.

[71] Offical SPIN Website, [Online]. Available: http://spinroot.com/. [Accessed 18 May 2016].

[72] M. Ben-Ari, "A primer on model checking," *ACM Inroads*, pp. 40-47, Mar 2010.

[73] J. Rushby, "Formal methods and their role in the certification of critical systems," in *Safety and reliability of software based systems*, London, Springer-Verlag, 1997, pp. 1-42.

[74] E. M. Clarke and E. A. Emerson, "Design and synthesis of synchronization skeletons using branching-time temporal logic," in *Logic of programs, workshop*, London, Springer-Verlag, 1981, pp. 52-71.

[75] C. Baier and J.-P. Katoen, Principles of model checking, Cambridge: The MIT Press, 2008.

[76] X. Tong, X. Wang, R. Wang, F. Huang, X. Dong, K. M. Hopkinson and G. Song, "The study of a regional decentralized peer-to-peer negotiation-based wide-area backup protection multi-agent system," *IEEE Transactions on Smart Grid*, vol. 4, no. 2, pp. 1197-1206, 2013.

[77] K. Hopkinson, X. Wang, R. Giovanini, J. Thorp, K. Birman and D. Coury, "EPOCHS: a platform for agent-based electric power and communication simulation built from commercial off-the-shelf components," *IEEE Transactions on Power Systems*, vol. 21, no. 2, pp. 548-558, 2006.

[78] R. Nagel, "Unraveling in guessing games: an experimental study," *The American Economic Review*, vol. 85, no. 5, pp. 1313-1326, 1995.

[79] B. Schneier, "Drugs: sports' prisoner's dilemma," 10 August 2006. [Online]. Available: https://www.schneier.com/essays/archives/2006/08/drugs_sports_prisone.html. [Accessed 15 March 2016].

[80] S. J. Brams, Biblical games, Cambridge: MIT Press, 2003.

[81] P. Talwalkar, "Game theory in Numb3rs: hide and seek," 2 December 2009. [Online]. Available: http://mindyourdecisions.com/blog/2009/12/02/game-theory-in-numb3rs-hide-and-seek. [Accessed 15 March 2016].

[82] A. Rubinstein, "Experience from a course in game theory: pre and post-class problem sets as a didactic device," October 1999. [Online]. Available: http://arielrubinstein.tau.ac.il/99/gt100.html. [Accessed 15 March 2016].

[83] K. Basu, "The traveler's dilemma: paradoxes of rationality in game theory," *The American Economic Review*, vol. 84, no. 2, pp. 391-395, 1994.

[84] The Princess Bride film, "The "battle of wits" scene," [Online]. Available: https://www.youtube.com/watch?v=U_eZmEiyTo0. [Accessed 15 March 2016].

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
15-09-2016	Doctoral Dissertation	Jan 2013 – Sep 2016

TITLE AND SUBTITLE
Improving the Cybersecurity of Cyber-Physical Systems Through Behavioral Game Theory and Model Checking in Practice and in Education

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Hamman, Seth T., Mr., Civ.

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way, Building 640
WPAFB OH 45433-8865

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT-ENG-DS-16-S-010

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Intentionally left blank

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
DISTRUBTION STATEMENT A. APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES
This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT
This dissertation presents automated methods based on behavioral game theory and model checking to improve the cybersecurity of cyber-physical systems (CPSs) and advocates teaching certain foundational principles of these methods to cybersecurity students. First, it encodes behavioral game theory's concept of level-k reasoning into an integer linear program that models a newly defined security Colonel Blotto game. This approach is designed to achieve an efficient allocation of scarce protection resources by anticipating attack allocations. A human subjects experiment based on a CPS infrastructure demonstrates its effectiveness. Next, it rigorously defines the term adversarial thinking, one of cybersecurity education's most important and elusive learning objectives, but for which no proper definition exists. It spells out what it means to "think like a hacker" by examining the characteristic thought processes of hackers through the lens of Sternberg's triarchic theory of intelligence. Next, a classroom experiment demonstrates that teaching basic game theory concepts to cybersecurity students significantly improves their strategic reasoning abilities. Finally, this dissertation applies the SPIN model checker to an electric power protection system and demonstrates a straightforward and effective technique for rigorously characterizing the degree of fault tolerance of complex CPSs, a key step in improving their defensive posture.

15. SUBJECT TERMS
behavioral game theory, cyber-physical systems, cybersecurity, education, SPIN model checker

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Kenneth M. Hopkinson, AFIT/ENG
U	U	U	UU	117	19b. TELEPHONE NUMBER *(Include area code)* (937) 255-3636, x4579 kenneth.hopkinson@afit.edu

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18